Measuring and Improving the Use of Graph Information in Graph Neural Networks

HOU, Yifan

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
June 2020

Thesis Assessment Committee

Professor LUI Chi Shing John (Chair)
Professor YANG Ming Chang (Thesis Supervisor)
Professor CHENG James (Committee Member)
Professor WU Hejun (External Examiner)

Abstract of thesis entitled:
Measuring and Improving the Use of Graph Information in Graph Neural Networks
Submitted by HOU, Yifan
for the degree of Master of Philosophy in Computer Science and Engineering
at The Chinese University of Hong Kong in June 2020

Representation learning on graphs, also called graph embedding, has demonstrated its significant impact on a series of machine learning applications such as classification, prediction and recommendation. Graph neural networks (GNNs) have been widely used for representation learning on graph data. However, existing work has largely ignored the rich information contained in the properties (or attributes) of both nodes and edges of graphs in modern applications, e.g., those represented by property graphs. Besides, there is limited understanding on how much performance GNNs actually gain from graph data.

This thesis we have two main contributions. First, we propose PGE, a graph neural networks framework that incorporates both node and edge properties into the graph embedding procedure. PGE uses node clustering to assign biases to differentiate neighbors of a node and leverages multiple data-driven matrices to aggregate the property information of neighbors sampled based on a biased strategy. PGE adopts the popular inductive model for neighborhood aggregation. Second, we introduce a context-surrounding GNN framework and propose two smoothness metrics to measure the quantity and quality of information obtained from graph data. A new GNN model, called CS-GNN, is then designed to improve the use of graph information based on the smoothness values of a graph. CS-GNN is shown to achieve better performance than existing methods in different types of real graphs.

摘要：

图上的表示学习（也称为图嵌入）已证明其对一系列机器学习应用程序（如分类，预测和推荐）的重大影响。图神经网络（GNN）已被广泛用于图数据的表示学习。然而，现有的工作在很大程度上忽略了现代应用中图的节点和边的属性中所包含的丰富信息，例如属性图。此外，对GNN实际上从图形数据中获得多少性能的理解还很有限。

本论文我们有两个主要贡献。首先，我们提出PGE，一种图神经网络框架，该框架将节点和边的属性都结合到了图嵌入过程中。PGE使用节点聚类来分配重要性，以区分节点的邻居，并利用多个参数矩阵来汇总基于重要性策略采样的邻居的属性信息。PGE采用流行的归纳模型进行邻域聚合。其次，我们引入了一个背景-环境的GNN框架（CS-GNN），并提出了两个平滑度度量来度量从图数据中获得的信息的数量和质量。然后设计一种称为CS-GNN的新GNN模型，以基于图形的平滑度值来改善图信息的使用。在不同类型的真实图数据中，CS-GNN被证明比现有方法具有更好的性能。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Backgrounds

Graphs are powerful data structures that allow us to easily express various relationships (i.e., edges) between objects (i.e., nodes), which are ubiquitous today due to the flexibility of using graphs to model data in a wide spectrum of applications. In recent years, more and more machine learning applications conduct classification or prediction based on graph data [7, 39, 22, 20], such as classifying protein's functions in biological graphs, understanding the relationship between users in online social networks, and predicting purchase patterns in buyers-products-sellers graphs in online e-commerce platforms. However, it is not easy to directly make use of the structural information of graphs in these applications as graph data are high-dimensional and non-Euclidean. On the other hand, considering only graph statistics such as degrees [6], kernel functions [19], or local neighborhood structures [32] often provides limited information and hence affects the accuracy of classification/prediction.

Representation learning methods [5] attempt to solve the above-mentioned problem by constructing an embedding for each node in a graph, i.e., a mapping from a node to a low-dimensional Euclidean space as vectors, which uses geometric metrics (e.g., Euclidean distance) in the embedding space to represent the structural information. Such graph embeddings [20, 22] have achieved good performance for classification/prediction on *plain graphs* (i.e., graphs with only the pure topology, without node/edge labels and properties).

Graph neural network (GNN) is one kind of representative method in representation learning. In recent years, extensive studies have been conducted on GNNs for tasks such as node classification and link predication. GNNs utilize the relationship information in graph data and significant improvements over traditional methods

have been achieved on benchmark datasets [29, 23, 55, 60]. Such breakthrough results have led to the exploration of using GNNs and their variants in different areas such as computer vision [49, 35], natural language processing [42, 62], chemistry [15], biology [18], and social networks [58]. Thus, understanding why GNNs can outperform traditional methods that are designed for Euclidean data is important. Such understanding can help us analyze the performance of existing GNN models and develop new GNN models for different types of graphs.

1.2 Rich Property Information on Graphs and PGE Framework

In practice, most graphs in real-world do not only contain the topology information, but also contain labels and *properties* (also called *attributes*) on the entities (i.e., nodes) and relationships (i.e., edges). For example, in the companies that we collaborate with, most of their graphs (e.g., various graphs related to products, buyers and sellers from an online e-commerce platform; mobile phone call networks and other communication networks from a service provider) contain rich node properties (e.g., user profile, product details) and edge properties (e.g., transaction records, phone call details). We call such graphs as **property graphs**. Existing methods [43, 21, 44, 10, 57, 27, 23] have not considered to take the rich information carried by both nodes and edges into the graph embedding procedure.

This thesis first studies the problem of property graph embedding. There are two main challenges. First, each node v may have many properties and it is hard to find which properties may have greater influence on v for a specific application. For example, consider the classification of papers into different topics for a citation graph where nodes represent papers and edges model citation relationships. Suppose that each node has two properties, "year" and "title". Apparently, the property "title" is likely to be more important for paper classification than the property "year". Thus, how to measure the influence of the properties on each node for different applications needs to be considered. Second, for each node v, its neighbors, as well as the connecting edges, may have different properties. How to measure the influences of both the neighbors and the connecting edges on v for different application poses another challenge. In the above example, for papers referencing a target paper, those with high citations should mean more to the target paper than those with low citations.

Among existing work, GCN [27] leverages node property information for node embedding generation, while GraphSAGE [23] extends GCN from a spectral method

to a spatial one. Given an application, GraphSAGE trains a weight matrix before embedding and then aggregates the property information of the neighbors of each node with the trained matrix to compute the node embedding. However, GraphSAGE does not differentiate neighbors with property dissimilarities for each node, but rather treats all neighbors equally when aggregating their property information. Moreover, GraphSAGE considers only node information and ignores edge directions and properties. Apart from the properties on nodes/edges, real-world graphs also have special structural features. For example, in social networks, nodes are often organized in the form of communities, where similar nodes are either neighbors due to the *homophily* feature [4, 3], or not direct neighbors but with similar structure due to the *structural equivalent* feature [17, 24, 61]. Thus, it is important to also consider structural features. For that, node2vec [21] learns node embeddings by combining two strategies, breadth-first random walk and depth-first random walk, to account for the homophily feature and structural equivalent feature. However, node2vec only utilizes these two structural features without considering any property information.

To address the limitations of existing methods, we propose a new framework, **PGE**, for property graph embedding. PGE applies a biased method to differentiate the influences of the neighbors and the corresponding connecting edges by incorporating both the topology and property information into the graph embedding procedure. The framework consists of three main steps: (1) *property-based node clustering* to classify the neighborhood of a node into similar and dissimilar groups based on their property similarity with the node; (2) *biased neighborhood sampling* to obtain a smaller neighborhood sampled according the bias parameters (which are set based on the clustering result), so that the embedding process can be more scalable; and (3) *neighborhood aggregation* to compute the final low dimensional node embeddings by aggregating the property information of sampled neighborhood with weight matrices trained with neural networks. We also analyze in details how the three steps work together to contribute to a good graph embedding and why our biased method (incorporating node and edge information) can achieve better embedding results than existing methods.

1.3 Understanding Graph Neural Networks and CS-GNN Framework

Besides that, we also discuss why GNN models can outperform existing Euclidean-based methods. One main reason is because rich information from the neighborhood of an object can be captured. GNNs collect neighborhood information with aggre-

gators [65], such as the *mean* aggregator that takes the mean value of neighbors' feature vectors [23], the *sum* aggregator that applies summation [15], and the *attention* aggregator that takes the weighted sum value [55]. Then, the aggregated vector and a node's own feature vector are combined into a new feature vector. After some rounds, the feature vectors of nodes can be used for tasks such as node classification. Thus, the performance improvement brought by graph data is highly related to the *quantity* and *quality* of the neighborhood information. To this end, we propose *two smoothness metrics on node features and labels to measure the quantity and quality of neighborhood information of nodes*. The metrics are used to analyze the performance of existing GNNs on different types of graphs.

In practice, not all neighbors of a node contain relevant information w.r.t. a specific task. Thus, neighborhood provides both positive information and negative disturbance for a given task. Simply aggregating the feature vectors of neighbors with manually-picked aggregators (i.e., users choose a type of aggregator for different graphs and tasks by trial or by experience) often cannot achieve optimal performance. To address this problem, we propose *a new model,* **CS-GNN** (context-surrounding graph neural network). *It uses the smoothness metrics to selectively aggregate neighborhood information to amplify useful information and reduce negative disturbance*. Our experiments validate the effectiveness of our two smoothness metrics and the performance improvements.

1.4 Contributions

Our main contributions are listed as follows:

- We propose PGE, which incorporates both topology and property information of nodes and edges into the graph embedding procedure leveraging and neighbors biased aggregation. In addition, it supports directed graph with various types of edges.
- We propose two graph smoothness metrics to help understand the use of graph information in GNNs. Based on that, we propose a new GNN model that improves the use of graph information using the smoothness values.

The thesis is organized as follows: In next chapter we briefly introduce related existing works, about graph embedding methods and representative GNN methods. The mechanism of GNNs is also included. In Chapter 3 and 4, we present the details of PGE framework and CS-GNN framework. Then we provide experimental results to verify our two frameworks in Chapter 5. In the last Chapter we conclude the thesis.

Chapter 2

Related Work

2.1 Graph Representation Learning

There are three main methods for graph embedding: *matrix factorization*, *random walk*, and *neighbors aggregation*.

For matrix factorization methods, [2, 8] use adjacency matrix to define and measure the similarity among nodes for graph embedding. HOPE [41] further preserves high-order proximities and obtains asymmetric transitivity for directed graphs. Another line of works utilize the random walk statistics to learn embeddings with the skip-gram model [37], which applies vector representation to capture word relationships.

The key idea of random walk is that nodes usually tend to co-occur on short random walks if they have similar embeddings [22]. DeepWalk [43] is the first to input random walk paths into a skip-gram model for learning node embeddings. node2vec [21] further utilizes biased random walks to improve the mapping of nodes to a low-dimensional space, while combining breadth-first walks and depth-first walks to consider graph homophily and structural equivalence. To obtain larger relationships, Walklets [44] involves *offset* to allow longer step length during a random walk, while HARP [10] makes use of graph preprocessing that compresses some nodes into one super-node to improve random walk.

According to [22], *matrix factorization* and *random walk* methods are *shallow embedding approaches* and have the following drawbacks. First, since the node embeddings are independent and there is no sharing of parameters or functions, these methods are not efficient for processing large graphs. Second, they do not consider node/edge properties. Third, as the embeddings are transductive and can only be generated during the training phrase, unseen nodes cannot be embedded with the

model being learnt so far.

To address (some of) the above problems, graph neural networks have been used to learn node embeddings. This method encodes nodes into vectors by compressing neighborhood information [25, 9, 57], which are introduced in details later. However, although this type of methods can share parameters, strictly speaking they are still transductive. Besides, they have performance bottlenecks for processing large graphs as the input dimensionality of auto-encoders is equal to the number of nodes. Several recent works [23, 28, 50, 27, 11] attempted to use only local neighborhood instead of the entire graph to learn node embeddings through neighbor aggregation, which can also consider property information on nodes. GCN [27] uses graph convolutional networks to learn node embeddings, by merging local graph structures and features of nodes to obtain embeddings from the hidden layers. GraphSAGE [23] is inductive and able to capture embeddings for unseen nodes through its trained auto-encoders directly. The advantage of neighborhood aggregation methods is that they not only consider the topology information, but also compute embeddings by aggregating property vectors of neighbors. However, existing neighborhood aggregation methods treat the property information of neighbors equally and fail to differentiate the influences of neighbors (and their connecting edges) that have different properties.

2.2 GNN Framework and Models

We use $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ to denote a graph, where $\mathcal{V}$ and $\mathcal{E}$ represent the set of nodes and edges of $\mathcal{G}$.[1] We use $e_{v,v'} \in \mathcal{E}$ to denote the edge that connects nodes v and v', and $\mathcal{N}_v = \{v' : e_{v,v'} \in \mathcal{E}\}$ to denote the set of neighbors of a node $v \in \mathcal{V}$. Each node $v \in \mathcal{V}$ has a feature vector $x_v \in \mathcal{X}$ with dimension d. Consider a node classification task, for each node $v \in \mathcal{V}$ with a class label y_v, the goal is to learn a representation vector h_v and a mapping function $f(\cdot)$ to predict the class label y_v of node v, i.e., $\hat{y}_v = f(h_v)$ where $\hat{y}_v$ is the predicted label.

GNNs are inspired by the Weisfeiler-Lehman test [59, 52], which is an effective method for graph isomorphism. Similarly, GNNs utilize a neighborhood aggregation scheme to learn a representation vector h_v for each node v, and then use neural networks to learn a mapping function $f(\cdot)$. Formally, consider the general GNN framework [23, 65, 60] in Table 2.1 with K rounds of neighbor aggregation. In each round, only the features of 1-hop neighbors are aggregated, and the framework consists of two functions, AGGREGATE and COMBINE. We initialize $h_v^{(0)} = x_v$.

[1]For the notation usages in PGE and CS-GNN, most of them are unified. But due to the large number of definitions, abuse of notation might occur between the two frameworks.

Table 2.1 Neighborhood aggregation schemes

Models	Aggregation and combination functions for round k $(1 \leq k \leq K)$
General GNN framework	$h_v^{(k)} = \text{COMBINE}^{(k)}\Big(\Big\{h_v^{(k-1)}, \text{AGGREGATE}^{(k)}\big(\{h_{v'}^{(k-1)} : v' \in \mathcal{N}_v\}\big)\Big\}\Big)$
GCN	$h_v^{(k)} = A\Big(\sum_{v' \in \mathcal{N}_v \cup \{v\}} \frac{1}{\sqrt{(\lvert\mathcal{N}_v\rvert+1)\cdot(\lvert\mathcal{N}_{v'}\rvert+1)}} \cdot W^{(k-1)} \cdot h_{v'}^{(k-1)}\Big)$
GraphSAGE	$h_v^{(k)} = A\Big(W^{(k-1)} \cdot \Big[h_v^{(k-1)} \big\vert\big\vert \text{AGGREGATE}\big(\{h_{v'}^{(k-1)}, v' \in \mathcal{N}_v\}\big)\Big]\Big)$
GAT	$h_{v_i}^{(k)} = A\Big(\sum_{v_j \in \mathcal{N}_{v_i} \cup \{v_i\}} a_{i,j}^{(k-1)} \cdot W^{(k-1)} \cdot h_{v_j}^{(k-1)}\Big)$

After K rounds of aggregation, each node $v \in \mathcal{V}$ obtains its representation vector $h_v^{(K)}$. We use $h_v^{(K)}$ and a mapping function $f(\cdot)$, e.g., a fully connected layer, to obtain the final results for a specific task such as node classification.

Many GNN models have been proposed. We introduce three representative ones: Graph Convolutional Networks (GCN) [29], GraphSAGE [23], and Graph Attention Networks (GAT) [55]. GCN merges the combination and aggregation functions, as shown in Table 2.1, where $A(\cdot)$ represents the activation function and W is a learnable parameter matrix. Different from GCN, GraphSAGE uses concatenation '||' as the combination function, which can better preserve a node's own information. Different aggregators (e.g., *mean*, *max pooling*) are provided in GraphSAGE. However, GraphSAGE requires users to choose an aggregator to use for different graphs and tasks, which may lead to sub-optimal performance. GAT addresses this problem by an attention mechanism that learns coefficients of neighbors for aggregation. With the learned coefficients $a_{i,j}^{(k-1)}$ on all the edges (including self-loops), GAT aggregates neighbors with a *weighted sum* aggregator. The attention mechanism can learn coefficients of neighbors in different graphs and achieves significant improvements over prior GNN models.

Chapter 3

PGE Framework

We use $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{P}, \mathcal{L}\}$ to denote a property graph , where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. $\mathcal{P}$ is the set of all properties and $\mathcal{P} = \mathcal{P}_\mathcal{V} \cup \mathcal{P}_\mathcal{E}$, where $\mathcal{P}_\mathcal{V} = \bigcup_{v \in \mathcal{V}} \{p_v\}$, $\mathcal{P}_\mathcal{E} = \bigcup_{e \in \mathcal{E}} \{p_e\}$, and p_v and p_e are the set of properties of node v and edge e, respectively. $\mathcal{L} = \mathcal{L}_\mathcal{V} \cup \mathcal{L}_\mathcal{E}$ is the set of labels, where $\mathcal{L}_\mathcal{V}$ and $\mathcal{L}_\mathcal{E}$ are the sets of node and edge labels, respectively. We use $\mathcal{N}_v$ to denote the set of neighbors of node $v \in \mathcal{V}$, i.e., $\mathcal{N}_v = \{v' : (v, v') \in \mathcal{E}\}$. In the case that $\mathcal{G}$ is directed, we may further define $\mathcal{N}_v$ as the set of in-neighbors and the set of out-neighbors, though in this paper we abuse the notation a bit and do not use new notations such as $\mathcal{N}_v^{in}$ and $\mathcal{N}_v^{out}$ for simplicity of presentation, as the meaning should be clear from the context.

The property graph model is general and can represent other popular graph models. If we set $\mathcal{P} = \emptyset$ and $\mathcal{L} = \emptyset$, then $\mathcal{G}$ becomes a *plain graph*, i.e., a graph with only the topology. If we set $\mathcal{P}_\mathcal{V} = \mathcal{A}$, $\mathcal{P}_\mathcal{E} = \emptyset$, and $\mathcal{L} = \emptyset$, where $\mathcal{A}$ is the set of node attributes, then $\mathcal{G}$ becomes an *attributed graph*. If we set $\mathcal{L} = \mathcal{L}_\mathcal{V}$, $\mathcal{P} = \emptyset$, and $\mathcal{L}_\mathcal{E} = \emptyset$, then $\mathcal{G}$ is a *labeled graph*.

3.1 Problem Definition

The main focus of PGE is to utilize both topology and property information in the embedding learning procedure to improve the results for different applications. Given a property graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{P}, \mathcal{L}\}$, we define the similarity between two nodes $v_i, v_j \in \mathcal{V}$ as $s_\mathcal{G}(v_i, v_j)$. The similarity can be further decomposed into two parts, $s_\mathcal{G}(v_i, v_j) = l(s_\mathcal{P}(v_i, v_j), s_\mathcal{T}(v_i, v_j))$, where $s_\mathcal{P}(v_i, v_j)$ is the property similarity and $s_\mathcal{T}(v_i, v_j)$ is the topology similarity between v_i and v_j, and $l(\cdot, \cdot)$ is a non-negative mapping.

The *embedding* of node $v \in \mathcal{V}$ is denoted as $\mathbf{z}_v$, which is a vector obtained by an

encoder ENC$(v) = \mathbf{z}_v$. Our objective is to find the optimal ENC$(\cdot)$, which minimizes the gap $\sum_{v_i,v_j \in \mathcal{V}} ||s_{\mathcal{G}}(v_i, v_j) - \mathbf{z}_{v_i}^\top \mathbf{z}_{v_j}|| = \sum_{v_i,v_j \in \mathcal{V}} ||l(s_{\mathcal{P}}(v_i, v_j), s_{\mathcal{T}}(v_i, v_j)) - \mathbf{z}_{v_i}^\top \mathbf{z}_{v_j}||$.

From the above problem definition, it is apparent that only considering the topology similarity $s_{\mathcal{T}}(v_i, v_j)$, as the traditional approaches do, cannot converge to globally optimal results. In addition, given a node v and its neighbors v_i, v_j, the property similarity $s_{\mathcal{P}}(v, v_i)$ can be very different from $s_{\mathcal{P}}(v, v_j)$. Thus, in the PGE framework, we use both topology similarity and property similarity in learning the node embeddings.

3.2 The Three Steps of PGE

The PGE framework consists of three major steps as follows.

- Step 1: **Property-based Node Clustering.** We cluster nodes in $\mathcal{G}$ based on their properties to produce k clusters $\mathcal{C}=\{C_1, C_2, ..., C_k\}$. A standard clustering algorithm such as K-Means [34] or DBSCAN [16] can be used for this purpose, where each node to be clustered is represented by its property vector (note that graph topology information is not considered in this step).
- Step 2: **Biased Neighborhood Sampling.** To combine the influences of property information and graph topology by $l(\cdot, \cdot)$, we conduct biased neighborhood sampling based on the results of clustering in Step 1. To be specific, there are two phases in this step: (1) For each neighbor $v' \in \mathcal{N}_v$, if v' and v are in the same cluster, we assign a *bias* b_s to v' to indicate that they are similar; otherwise we assign a different *bias* b_d to v' instead to indicate that they are dissimilar. (2) We normalize the assigned biases on $\mathcal{N}_v$, and then sample $\mathcal{N}_v$ according to the normalized biases to obtain a fixed-size sampled neighborhood $\mathcal{N}_v^s$.
- Step 3: **Neighborhood Aggregation.** Based on the sampled neighbors $\mathcal{N}_v^s$ in Step 2, we aggregate their property information to obtain $\mathbf{z}_v$ by multiplying the weight matrices that are trained with neural networks.

In the following three sub-sections, we discuss the purposes and details of each of the above three steps.

3.2.1 Property-based Node Clustering

The purpose of Step 1 is to classify $\mathcal{N}_v$ into two types for each node v based on their node property information, i.e., those similar to v or dissimilar to v. If v and its

neighbor $v' \in \mathcal{N}_v$ are in the same cluster, we will regard v' as a similar neighbor of v. Otherwise, v' is dissimilar to v.

Due to the high dimensionality and sparsity of properties (e.g., property values are often texts but can also be numbers and other types), which also vary significantly from datasets to datasets, it is not easy to classify the neighborhood of each node into similar and dissimilar groups while maintaining a unified global standard for classifying the neighborhood of all nodes. For example, one might attempt to calculate the property similarity between v and each of v's neighbors, for all $v \in \mathcal{V}$, and then set a threshold to classify the neighbors into similar and dissimilar groups. However, different nodes may require a different threshold and their similarity ranges can be very different. Moreover, each node's neighborhood may be classified differently and as we will show later, the PGE framework actually uses the 2-hop neighborhood while this example only considers the 1-hop neighborhood. Thus, we need a unified global standard for the classification. For this purpose, clustering the nodes based on their properties allows all nodes to be classified based on the same global standard. For example, the 1-hop neighbors and the 2-hop neighbors of a node v are classified in the same way based on whether they are in the same cluster as v.

3.2.2 Biased Neighborhood Sampling

Many real-world graphs have high-degree nodes, i.e., these nodes have a large number of neighbors. It is inefficient and often unnecessary to consider all the neighbors for neighborhood aggregation in Step 3. Therefore, we use the biases b_s and b_d to derive a sampled neighbor set $\mathcal{N}_v^s$ with a fixed size for each node v. As a result, we obtain a sampled graph $\mathcal{G}^s = \{\mathcal{V}, \mathcal{E}^s\}$, where $\mathcal{E}^s = \{(v, v') : v' \in \mathcal{N}_v^s\}$. Since the biases b_s and b_d are assigned to the neighbors based on the clustering results computed from the node properties, $\mathcal{G}^s$ contains the topology information of $\mathcal{G}$ while it is constructed based on the node property information. Thus, Step 2 is essentially a mapping $l(\cdot, \cdot)$ that fuses $s_{\mathcal{P}}(v, v')$ and $s_{\mathcal{T}}(v, v')$.

The biases b_s and b_d are the un-normalized possibility of selecting neighbors from dissimilar and similar clusters, respectively. The value of b_s is set to 1, while b_d can be varied depending on the probability (greater b_d means higher probability) that dissimilar neighbors should be selected into $\mathcal{G}^s$. We will analyze the effects of the bias values in Section 3.5 and verify by experimental results in Section 5.1.3. The size of $\mathcal{N}_v^s$ is set to 25 by default following GraphSAGE [23] (also for fair comparison in our experiments). The size 25 was found to be a good balance point in [23] as a larger size will significantly increase the model computation time, though in the case of PGE as it differentiates neighbors, using a sampled neighborhood could achieve a

better quality of embedding than using the full neighborhood.

3.2.3 Neighborhood Aggregation

The last step is to learn the low dimensional embedding with $\mathcal{G}^s = \{\mathcal{V}, \mathcal{E}^s\}$. We use neighborhood aggregation to learn the function ENC($\cdot$) for generating the node embeddings. For each node, we select its neighbors within two hops to obtain $\mathbf{z}_v$ by the following equations:

$$\mathbf{z}_v = \sigma(W^1 \cdot \mathrm{A}(\mathbf{z}^1_v, \sum_{v' \in \mathcal{N}^s_v} \mathbf{z}^1_{v'}/|\mathcal{N}^s_v|)),$$

$$\mathbf{z}^1_{v'} = \sigma(W^2 \cdot \mathrm{A}(p_{v'}, \sum_{v'' \in \mathcal{N}^s_{v'}} p_{v''}/|\mathcal{N}^s_{v'}|)),$$

where p_v is the original property vector of node v, $\sigma(\cdot)$ is the nonlinear activation function and A($\cdot$) is the *concatenate* operation. We use two weight matrices W^1 and W^2 to aggregate the node property information of v's one-hop neighbors and two-hop neighbors.

The matrix W^i is used to assign different weights to different properties because aggregating (e.g., taking mean value) node property vectors directly cannot capture the differences between properties, but different properties contribute to the embedding in varying degrees. Also, the weight matrix is data-driven and should be trained separately for different datasets and applications, since nodes in different graphs have different kinds of properties. The weight matrices are pre-trained using Adam SGD optimizer [26], with a loss function defined for the specific application, e.g., for node classification, we use binary cross entropy loss (multi-labeled); for link prediction, we use cross entropy loss with negative sampling.

3.3 Support of Edge Direction and Properties

The sampled graph $\mathcal{G}^s$ does not yet consider the edge direction and edge properties. To include edge properties, we follow the same strategy as we do on nodes. If edges are directed, we consider in-edges and out-edges separately. We cluster the edges into k^e clusters $\mathcal{C}^e = \{C^e_1, C^e_2, ..., C^e_{k^e}\}$. Then, we train $2 \cdot k^e$ matrices, $\{W^1_1, W^1_2, ..., W^1_{k^e}\}$ and $\{W^2_1, W^2_2, ..., W^2_{k^e}\}$, to aggregate node properties for k^e types of edges for the 2-hop neighbors. Finally, we obtain $\mathbf{z}_v$ by the following equations:

$$\mathbf{z}_v = \sigma\Big(\mathrm{A}\big(W^1_0 \cdot \mathbf{z}^1_v, \mathrm{A}_{C^e_i \in \mathcal{C}^e}(W^1_i \cdot \mathbb{E}_{v' \in \mathcal{N}^s_v \,\&\, (v,v') \in C^e_i}[\mathbf{z}^1_{v'}])\big)\Big), \tag{1}$$

Algorithm 1 Property Graph Embedding (PGE)

Input: A Property Graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{P}\}$; biases b_d and b_s; the size of sampled neighborhood $|\mathcal{N}_v^s|$; weight matrices $\{W_1^1, W_2^1, ..., W_{k^e}^1\}$ and $\{W_1^2, W_2^2, ..., W_{k^e}^2\}$

Output: Low-dimensional representation vectors $\mathbf{z}_v$, $\forall v \in \mathcal{V}$

Clustering $\mathcal{V}$, $\mathcal{E}$, and obtain $\mathcal{C}$ and $\mathcal{C}^e$ based on $\mathcal{P}$; ▷ step 1
for all $v \in \mathcal{V}$ **do** ▷ step 2
  for all $v' \in \mathcal{N}_v$ **do**
    Assign $b = b_d + (b_s - b_d) \cdot \sum_{C_i \in \mathcal{C}} \mathbb{I}\{v, v' \in C_i\}$ to v',
    where $\mathbb{I}\{v, v' \in C_i\} = 1$ if $v, v' \in C_i$ and 0 otherwise;
  end for
  Sample $|\mathcal{N}_v^s|$ neighbors with bias b;
end for
for all $v \in \mathcal{V}$ **do** ▷ step 3
  Compute $\mathbf{z}_v^1$ with Equation (2);
end for
for all $v \in \mathcal{V}$ **do**
  Compute $\mathbf{z}_v$ with Equation (1);
end for

$$\mathbf{z}_{v'}^1 = \sigma\Big(\mathrm{A}\big(W_0^2 \cdot p_{v'}, \mathrm{A}_{C_i^e \in \mathcal{C}^e}(W_i^2 \cdot \mathbb{E}_{v'' \in \mathcal{N}_{v'}^s \,\&\, (v', v'') \in C_i^e}[p_{v''}])\big)\Big). \tag{2}$$

Note that $|\mathcal{C}^e|$ should not be too large as to avoid high-dimensional vector operations. Also, if $|\mathcal{C}^e|$ is too large, some clusters may contain only a few elements, leading to under-fitting for the trained weight matrices. Thus, we set $|\mathcal{C}^e|$ as a fixed small number.

3.4 The Algorithm

Algorithm 1 presents the overall procedure of computing the embedding vector $\mathbf{z}_v$ of each node $v \in \mathcal{V}$. The algorithm follows exactly the three steps that we have described in Section 3.2.

3.5 An Analysis of PGE

In this section, we present a detailed analysis of PGE. In particular, we analyze why the biased strategy used in PGE can improve the embedding results. We also

discuss how the bias values b_d and b_s and edge information affect the embedding performance.

3.5.1 The Efficacy of the Biased Strategy

One of the main differences between PGE and GraphSAGE [23] is that neighborhood sampling in PGE is biased (i.e., neighbors are selected based on probability values defined based on b_d and b_s), while GraphSAGE's neighborhood sampling is unbiased (i.e., neighbors are sampled with equal probability). We analyze the difference between the biased and the unbiased strategies in the subsequent discussion.

We first argue that neighborhood sampling is a special case of random walk. For example, if we set the walk length to 1 and perform 10 times of walk, the strategy can be regarded as 1-hop neighborhood sampling with a fixed size of 10. Considering that the random walk process in each step follows an i.i.d. process for all nodes, we define the biased strategy as a $|\mathcal{V}| \times |\mathcal{V}|$ matrix $\mathbf{P}$, where $\mathbf{P}_{i,j}$ is the probability that node v_i selects its neighbor v_j in the random walk. If two nodes v_i and v_j are not connected, then $\mathbf{P}_{i,j} = 0$. Similarly, we define the unbiased strategy $\mathbf{Q}$, where all neighbors of any node have the same probability to be selected. We also assume that there exists an optimal strategy $\mathbf{B}$, which gives the best embedding result for a given application.

A number of works [21, 44, 10] have already shown that adding preference on similar and dissimilar neighbors during random walk can improve the embedding results, based on which we have the following statement: *for a biased strategy $\boldsymbol{P}$, if $||\boldsymbol{B}-\boldsymbol{P}||_1 < ||\boldsymbol{B}-\boldsymbol{Q}||_1$, where $\boldsymbol{B} \neq \boldsymbol{Q}$, then $\boldsymbol{P}$ has a positive influence on improving the embedding results.*

Thus, to verify the efficacy of PGE's biased strategy, we need to show that our strategy $\mathbf{P}$ satisfies $||\mathbf{B} - \mathbf{P}||_1 \leq ||\mathbf{B} - \mathbf{Q}||_1$. To do so, we show that b_d and b_s can be used to adjust the strategy $\mathbf{P}$ to get closer to $\mathbf{B}$ (than $\mathbf{Q}$).

Assume that nodes are classified into k clusters $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ based on the property information $\mathcal{P}_\mathcal{V}$. For the unbiased strategy, the expected similarity of two nodes $v, v' \in \mathcal{V}$ for each random walk step is:

$$\mathbb{E}[s_{\mathcal{G}(v,v')}] = \frac{\sum_{v\in\mathcal{V}} \sum_{v_i\in\mathcal{N}_v} s_\mathcal{G}(v, v_i)}{|\mathcal{E}|}.$$

The expectation of two nodes' similarity for each walk step in our biased strategy is:

$$\mathbb{E}[s_{\mathcal{G}(v,v')}] = \frac{\sum_{v\in\mathcal{V}} \sum_{v_i\in\mathcal{N}_v\cap C_v} n_s(v) \cdot s_\mathcal{G}(v, v_i)}{\frac{|\mathcal{E}|}{k}}$$

$$+\frac{\sum_{v\in\mathcal{V}}\sum_{v_j\in\mathcal{N}_v\cap(C_v)^c} n_d(v)\cdot s_{\mathcal{G}}(v,v_j)}{\frac{|\mathcal{E}|\cdot(k-1)}{k}}, \tag{3}$$

where $n_s(v)$ and $n_d(v)$ are the normalized biases of b_s and b_d for node v respectively, C_v is the cluster that contains v, and $(C_v)^c = \mathcal{C}\backslash\{C_v\}$. Since only connected nodes are to be selected in a random walk step, the normalized biases $n_s(v)$ and $n_d(v)$ can be derived by

$$n_s(v) = \frac{b_s}{b_d\cdot\sum_{v'\in\mathcal{N}_v}\mathbb{I}\{v'\in C_v\} + b_s\cdot\sum_{v'\in\mathcal{N}_v}\mathbb{I}\{v'\in(C_v)^c\}},$$

and

$$n_d(v) = n_s(v)\times\frac{b_d}{b_s}.$$

Consider Equation (3), if we set $b_d = b_s$, which means $n_d(v) = n_s(v)$, then it degenerates to the unbiased random walk strategy. But if we set b_d and b_s differently, we can adjust the biased strategy to either (1) select more dissimilar neighbors by assigning $b_d > b_s$ or (2) select more similar neighbors by assigning $b_s > b_d$.

Assume that the clustering result is not trivial, i.e., we obtain at least more than 1 cluster, we can derive that

$$\frac{\sum_{C_i\in\mathcal{C}}\sum_{v,v'\in C_i} s_{\mathcal{P}}(v,v')}{\frac{1}{2}\sum_{C_i\in\mathcal{C}}|C_i|\cdot(|C_i|-1)} > \frac{\sum_{v,v'\in\mathcal{V}} s_{\mathcal{P}}(v,v')}{\frac{1}{2}|V|\cdot(|V|-1)}.$$

Since $l(\cdot,\cdot)$ is a non-negative mapping with respect to $s_{\mathcal{P}}(v,v')$, we have

$$\frac{\sum_{C_i\in\mathcal{C}}\sum_{v,v'\in C_i} s_{\mathcal{G}}(v,v')}{\frac{1}{2}\sum_{C_i\in\mathcal{C}}|C_i|\cdot(|C_i|-1)} > \frac{\sum_{v,v'\in\mathcal{V}} s_{\mathcal{G}}(v,v')}{\frac{1}{2}|V|\cdot(|V|-1)} \tag{4}.$$

Equation (4) shows that the similarity $s_{\mathcal{G}}(v,v')$ is higher if v and v' are in the same cluster. Thus, based on Equations (3) and (4), we conclude that parameters b_d and b_s can be used to select similar and dissimilar neighbors.

Next, we consider the optimal strategy $\mathbf{B}$ for 1-hop neighbors, where $\mathbf{B}_{i,j} = \mathbb{I}\{v_j\in\mathcal{N}_{v_i}\}\cdot b^*_{v_i,v_j}$, and $b^*_{v_i,v_j}$ is the normalized optimal bias value for $\mathbf{B}_{i,j}$. Similarly, the unbiased strategy is $\mathbf{Q}_{i,j} = \mathbb{I}\{v_j\in\mathcal{N}_{v_i}\}\cdot\frac{1}{|\mathcal{N}_{v_i}|}$. Thus, we have

$$||\mathbf{B}-\mathbf{Q}||_1 = \sum_{v_i\in\mathcal{V}}\sum_{v_j\in\mathcal{V}}\left|b^*_{v_i,v_j} - \frac{1}{|\mathcal{N}_{v_i}|}\right|.$$

For our biased strategy, $\mathbf{P}_{i,j} = \mathbb{I}\{v_j\in\mathcal{N}_{v_i}\cap C_{v_i}\}\cdot n_s(v) + \mathbb{I}\{v_j\in\mathcal{N}_{v_i}\cap(C_{v_i})^c\}\cdot n_d(v)$. There exist b_s and b_d that satisfy $\sum_{v_i\in\mathcal{V}}\sum_{v_j\in\mathcal{V}}\left|b^*_{v_i,v_j} - \frac{1}{|\mathcal{N}_{v_i}|}\right| \geq \sum_{v_i\in\mathcal{V}}\sum_{v_j\in\mathcal{V}}\left|b^*_{v_i,v_j} -\right.$

$\mathbb{I}\{v_j \in \mathcal{N}_{v_i} \cap C_{v_i}\} \cdot n_s(v) - \mathbb{I}\{v_j \in \mathcal{N}_{v_i} \cap (C_{v_i})^c\} \cdot n_d(v)\Big|$, where strict inequality can be derived if $b_d \neq b_s$. Thus, $||\mathbf{B} - \mathbf{P}||_1 < ||\mathbf{B} - \mathbf{Q}||_1$ if we set proper values for b_s and b_d (we discuss the bias values in Section 3.5.2). Without loss of generality, the above analysis can be extended to the case of multi-hop neighbors.

3.5.2 The Effects of the Bias Values

Next we discuss how to set the proper values for the biases b_s and b_d for neighborhood sampling. We also analyze the impact of the number of clusters on the performance of PGE.

For neighborhood aggregation in Step 3 of PGE, an accurate embedding of a node v should be obtained by covering the whole connected component that contains v, where all neighbors within k-hops (k is the maximum reachable hop) should be aggregated. However, for a large graph, the execution time of neighborhood aggregation increases rapidly beyond 2 hops, especially for power-law graphs. For this reason, we trade accuracy by considering only the 2-hop neighbors. In order to decrease the accuracy degradation, we can enlarge the change that a neighbor can contribute to the embedding $\mathbf{z}_v$ by selecting dissimilar neighbors within the 2-hops, which we elaborate as follows.

Consider a node $v \in \mathcal{V}$ and its two neighbors $v_i, v_j \in \mathcal{N}_v$, and assume that $\mathcal{N}_{v_i} = \mathcal{N}_{v_j}$ but $|p_v - p_{v_i}| < |p_v - p_{v_j}|$. Thus, we have $s_{\mathcal{T}}(v, v_i) = s_{\mathcal{T}}(v, v_j)$ and $s_{\mathcal{P}}(v, v_i) > s_{\mathcal{P}}(v, v_j)$. Since $l(\cdot, \cdot)$ is a non-negative mapping, we also have $s_{\mathcal{G}}(v, v_i) > s_{\mathcal{G}}(v, v_j)$. Based on the definitions of $\mathbf{z}_v$ and $\mathbf{z}^1_{v'}$ given in Section 3.2.3, by expanding $\mathbf{z}^1_{v'}$ in $\mathbf{z}_v$, we obtain

$$\mathbf{z}_v = \sigma\Bigg(W^1 \cdot \mathrm{A}\Big(\mathbf{z}^1_v, \sum_{v' \in \mathcal{N}^s_v} \sigma\big(W^2 \cdot \mathrm{A}(p_{v'}, \sum_{v'' \in \mathcal{N}^s_{v'}} p_{v''}/|\mathcal{N}^s_{v'}|)\big)/|\mathcal{N}^s_v|\Big)\Bigg). \tag{5}$$

Equation (5) aggregates the node property vector p_v (which is represented within $\mathbf{z}^1_v$) and the property vectors of v's 2-hop neighbors to obtain the node embedding $\mathbf{z}_v$. This procedure can be understood as transforming from $s_{\mathcal{P}}(v, v')$ to $s_{\mathcal{G}}(v, v')$. Thus, a smaller $s_{\mathcal{P}}(v, v')$ is likely to contribute a more significant change to $\mathbf{z}_v$. With Equation (5), if $|p_v - p_{v_i}| < |p_v - p_{v_j}|$, we obtain $||\mathbf{z}^1_v - \mathbf{z}^1_{v_i}||_1 < ||\mathbf{z}^1_v - \mathbf{z}^1_{v_j}||_1$. Then, for the embeddings, we have $||\mathbf{z}_v - \mathbf{z}_{v_i}||_1 < ||\mathbf{z}_v - \mathbf{z}_{v_j}||_1$. Since v and v_i, as well as v and v_j, have mutual influence on each other, we conclude that for fixed-hop neighborhood aggregation, the neighbors with greater dissimilarity can contribute larger changes to the node embeddings. That is, for fixed-hop neighborhood aggregation, we should set $b_d > b_s$ for better embedding results, which is also validated in our experiments.

Apart from the values of b_d and b_s, the number of clusters obtained in Step 1 of PGE may also affect the quality of the node embeddings. Consider a random graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{P}\}$ with average degree $|\mathcal{E}|/|\mathcal{V}|$. Assume that we obtain k clusters from $\mathcal{G}$ in Step 1, then the average number of neighbors in $\mathcal{N}_v$ that are in the same cluster with a node v is $|\mathcal{N}_v|/k = (|\mathcal{E}|/|\mathcal{V}|)/k$. If k is large, most neighbors will be in different clusters from the cluster of v. On the contrary, a small k means that neighbors in $\mathcal{N}_v$ are more likely to be in the same cluster as v. Neither an extremely large k or small k gives a favorable condition for node embedding based on the biased strategy because we will have either all dissimilar neighbors or all similar neighbors, which essentially renders the neighbors in-differentiable. Therefore, to ensure the efficacy of the biased strategy, the value of k should not fall into either of the two extreme ends. We found that a value of k close to the average degree is a good choice based on our experimental results.

3.5.3 Incorporating Edge Properties

In addition to the biased values and the clustering number, the edge properties can also bring significant improvements on the embedding results. Many real-world graphs such as online social networks have edge properties like "positive" and "negative". Consider a social network $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{P}\}$ with two types of edges, $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$. Suppose that there is a node $v \in \mathcal{V}$ having two neighbors $v_i, v_j \in \mathcal{N}_v$, and these two neighbors have exactly the same property information $p_{v_i} = p_{v_j}$ and topology information $\mathcal{N}_{v_i} = \mathcal{N}_{v_j}$, but are connected to v with different types of edges, i.e., $(v, v_i) \in \mathcal{E}^+$ and $(v, v_j) \in \mathcal{E}^-$. If we only use Equation (5), then we cannot differentiate the embedding results of v_i and v_j ($\mathbf{z}_{v_i}$ and $\mathbf{z}_{v_j}$). This is because the edges are treated equally and the edge property information is not incorporated into the embedding results. In order to incorporate the edge properties, we introduce an extra matrix for each property. For example, in our case two additional matrices are used for the edge properties "positive" and "negative"; that is, referring to Section 3.3, we have $k^e = 2$ in this case. In the case of directed graphs, we further consider the in/out-neighbors separately with different weight matrices as we have discussed in Section 3.3.

Chapter 4

CS-GNN Framework

4.1 Graph Smoothness Metrics

GNNs usually contain an aggregation step to collect neighboring information and a combination step that merges this information with node features. We consider the *context* c_v of node v as the node's own information, which is initialized as the feature vector x_v of v[1]. We use s_v to denote the *surrounding* of v, which represents the aggregated feature vector computed from v's neighbors. Since the neighborhood aggregation can be seen as a convolution operation on a graph [13], we generalize the aggregator as weight linear combination, which can be used to express most existing aggregators. Then, we can re-formulate the general GNN framework as *a context-surrounding framework* with two mapping functions $f_1(\cdot)$ and $f_2(\cdot)$ in round k as:

$$c_{v_i}^{(k)} = f_1(c_{v_i}^{(k-1)}, s_{v_i}^{(k-1)}), \quad s_{v_i}^{(k-1)} = f_2(\sum\nolimits_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k-1)} \cdot c_{v_j}^{(k-1)}). \tag{1}$$

From equation (1), the key difference between GNNs and traditional neural-network-based methods for Euclidean data is that GNNs can integrate extra information from the surrounding of a node into its context. In graph signal processing [40], features on nodes are regarded as signals and it is common to assume that observations contain both noises and true signals in a standard signal processing problem [45]. Thus, we can decompose a context vector into two parts as $c_{v_i}^{(k)} = \breve{c}_{v_i}^{(k)} + \breve{n}_{v_i}^{(k)}$, where $\breve{c}_{v_i}^{(k)}$ is the true signal and $\breve{n}_{v_i}^{(k)}$ is the noise.

Theorem 1. *Assume that the noise $\breve{n}_{v_i}^{(k)}$ follows the same distribution for all nodes. If the noise power of $\breve{n}_{v_i}^{(k)}$ is defined by its variance σ^2, then the noise power of the*

[1]The notations in this chapter are summarized in Appendix B

surrounding input $\sum_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k-1)} \cdot c_{v_j}^{(k-1)}$ *is* $\sum_{v_j \in \mathcal{N}_{v_i}} (a_{i,j}^{(k-1)})^2 \cdot \sigma^2$.

Proof. We use the variance of $\breve{n}_{v_j}^{(k-1)}$ to measure the power of noise. Without loss of generality, we assume that the signal $\breve{c}_{v_j}^{(k-1)}$ is uncorrelated to the noise $\breve{n}_{v_j}^{(k-1)}$ and noise is random, with a mean of zero and constant variance. If we define

$$\mathrm{Var}\big(\breve{n}_{v_j}^{(k-1)}\big) = \mathbb{E}\big[\big(\breve{n}_{v_j}^{(k-1)}\big)^2\big] = \sigma^2,$$

then after the weight aggregation, the noise power of $\sum_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k-1)} \cdot c_{v_j}^{(k-1)}$ is

$$\begin{aligned}
\mathrm{Var}\Big(\sum\nolimits_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k-1)} \cdot \breve{n}_{v_j}^{(k-1)}\Big) &= \mathbb{E}\Big[\Big(\sum\nolimits_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k-1)} \cdot \breve{n}_{v_j}^{(k-1)}\Big)^2\Big] \\
&= \sum\nolimits_{v_j \in \mathcal{N}_{v_i}} \big(a_{i,j}^{(k-1)}\sigma\big)^2 \\
&= \sigma^2 \cdot \sum\nolimits_{v_j \in \mathcal{N}_{v_i}} \big(a_{i,j}^{(k-1)}\big)^2.
\end{aligned}$$

□

Theorem 1 shows that the surrounding input has less noise power than the context when a proper aggregator (i.e., coefficient $a_{i,j}^{(k-1)}$) is used. Specifically, the *mean* aggregator has the best denoising performance and the *pooling* aggregator (e.g., max-pooling) cannot reduce the noise power. For the *sum* aggregator, where all coefficients are equal to 1, the noise power of the surrounding input is larger than that of the context.

4.1.1 Feature Smoothness

We first analyze the information gain from the surrounding without considering the noise. In the extreme case when the context is the same as the surrounding input, the surrounding input contributes no extra information to the context. To quantify the information obtained from the surrounding, we present the following definition based on information theory.

Definition 2 (Information Gain from Surrounding)**.** *For normalized feature space* $\mathcal{X}_k = [0,1]^{d_k}$*, if* $\sum_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k)} = 1$*, the feature space of* $\sum_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k)} \cdot \breve{c}_{v_j}^{(k)}$ *is also in* $\mathcal{X}_k = [0,1]^{d_k}$*. The probability density function (PDF) of* $\breve{c}_{v_j}^{(k)}$ *over* $\mathcal{X}_k$ *is defined as* $C^{(k)}$*, which is the ground truth and can be estimated by nonparametric methods with a set of samples, where each sample point* $\breve{c}_{v_i}^{(k)}$ *is sampled with probability* $|\mathcal{N}_{v_i}|/2|\mathcal{E}|$*.*

Correspondingly, the PDF of $\sum_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k)} \cdot \breve{c}_{v_j}^{(k)}$ *is* $S^{(k)}$*, which can be estimated with a set of samples* $\{\sum_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k)} \cdot \breve{c}_{v_j}^{(k)}\}$*, where each point is sampled with probability* $|\mathcal{N}_{v_i}|/2|\mathcal{E}|$*. The information gain from the surrounding in round* k *can be computed by Kullback–Leibler divergence [30] as*

$$D_{KL}(S^{(k)}||C^{(k)}) = \int_{\mathcal{X}_k} S^{(k)}(\boldsymbol{x}) \cdot \log \frac{S^{(k)}(\boldsymbol{x})}{C^{(k)}(\boldsymbol{x})} d\boldsymbol{x}.$$

The Kullback–Leibler divergence is a measure of information loss when the context distribution is used to approximate the surrounding distribution [31]. Thus, we can use the divergence to measure the information gain from the surrounding into the context of a node. When all the context vectors are equal to their surrounding inputs, the distribution of the context is totally the same with that of the surrounding. In this case, the divergence is equal to 0, which means that there is no extra information that the context can obtain from the surrounding. On the other hand, if the context and the surrounding of a node have different distributions, the divergence value is strictly positive. Note that in practice, the ground-truth distributions of the context and surrounding signals are unknown. In addition, for learnable aggregators, e.g., the attention aggregator, the coefficients are unknown. Thus, we propose a metric λ_f to estimate the divergence. Graph smoothness [64] is an effective measure of the signal frequency in graph signal processing [45]. Inspired by that, we define the *feature smoothness* on a graph.

Definition 3 (*Feature Smoothness*)**.** *Consider the condition of the first round, where* $c_v^{(0)} = x_v$*, we define the feature smoothness* λ_f *over normalized space* $\mathcal{X} = [0,1]^d$ *as*

$$\lambda_f = \frac{\left|\left| \sum_{v \in \mathcal{V}} \left(\sum_{v' \in \mathcal{N}_v} (x_v - x_{v'}) \right)^2 \right|\right|_1}{|\mathcal{E}| \cdot d},$$

where $||\cdot||_1$ *is the Manhattan norm.*

According to Definition 3, a larger λ_f indicates that the feature signal of a graph has *higher frequency*, meaning that the feature vectors x_v and $x_{v'}$ are more likely dissimilar for two connected nodes v and v' in the graph. In other words, nodes with dissimilar features tend to be connected. Intuitively, for a graph whose feature sets have high frequency, the context of a node can obtain more information gain from its surrounding. This is because the PDFs (given in Definition 2) of the context and the surrounding have the same probability but fall in different places in space $\mathcal{X}$. Formally, we state the relation between λ_f and the information gain from

the surrounding in the following theorem. For simplicity, we let $\mathcal{X} = \mathcal{X}_0$, $d = d_0$, $C = C^{(0)}$ and $S = S^{(0)}$.

Theorem 4. *For a graph $\mathcal{G}$ with the set of features $\mathcal{X}$ in space $[0,1]^d$ and using the mean aggregator, the information gain from the surrounding $D_{KL}(S||C)$ is positively correlated to its feature smoothness λ_f, i.e., $D_{KL}(S||C) \sim \lambda_f$. In particular, $D_{KL}(S||C) = 0$ when $\lambda_f = 0$.*

Proof. For simplicity of presentation, let $\mathcal{X} = \mathcal{X}_0$, $d = d_0$, $C = C^{(0)}$ and $S = S^{(0)}$. For $D_{KL}(S||C)$, since the PDFs of C and S are unknown, we use a nonparametric way, histogram, to estimate the PDFs of C and S. Specifically, we uniformly divide the feature space $\mathcal{X} = [0,1]^d$ into r^d bins $\{H_1, H_2, ..., H_{r^d}\}$, whose length is $\frac{1}{r}$ and dimension is d. To simplify the use of notations, we use $|H_i|_C$ and $|H_i|_S$ to denote the number of samples that are in bin H_i. Thus, we have

$$
\begin{aligned}
D_{KL}(S||C) &\approx D_{KL}(\hat{S}||\hat{C}) \\
&= \sum_{i=1}^{r^d} \frac{|H_i|_S}{2|\mathcal{E}|} \cdot \log \frac{\frac{|H_i|_S}{2|\mathcal{E}|}}{\frac{|H_i|_C}{2|\mathcal{E}|}} \\
&= \frac{1}{2|\mathcal{E}|} \cdot \sum_{i=1}^{r^d} |H_i|_S \cdot \log \frac{|H_i|_S}{|H_i|_C} \\
&= \frac{1}{2|\mathcal{E}|} \cdot \Big(\sum_{i=1}^{r^d} |H_i|_S \cdot \log |H_i|_S - \sum_{i=1}^{r^d} |H_i|_S \cdot \log |H_i|_C \Big) \\
&= \frac{1}{2|\mathcal{E}|} \cdot \Big(\sum_{i=1}^{r^d} |H_i|_S \cdot \log |H_i|_S - \sum_{i=1}^{r^d} |H_i|_S \cdot \log \left(|H_i|_S + \Delta_i\right) \Big),
\end{aligned}
$$

where $\Delta_i = |H_i|_C - |H_i|_S$. Regard Δ_i as an independent variable, we consider the term $\sum_{i=1}^{r^d} |H_i|_S \cdot \log \left(|H_i|_S + \Delta_i\right)$ with second-order Taylor approximation at point 0 as

$$
\sum_{i=1}^{r^d} |H_i|_S \cdot \log \left(|H_i|_S + \Delta_i\right) \approx \sum_{i=1}^{r^d} |H_i|_S \cdot \Big(\log |H_i|_S + \frac{\ln 2}{|H_i|_S} \cdot \Delta_i - \frac{\ln 2}{2(|H_i|_S)^2} \cdot \Delta_i^2 \Big).
$$

Note that the numbers of samples for the context and the surrounding are the same, where we have

$$
\sum_{i=1}^{r^d} |H_i|_C = \sum_{i=1}^{r^d} |H_i|_S = 2 \cdot |\mathcal{E}|.
$$

Thus, we obtain

$$\sum_{i=1}^{r^d} \Delta_i = 0.$$

Therefore, the $D_{KL}(\hat{S}||\hat{C})$ can be written as

$$\begin{aligned}
D_{KL}(S||C) &\approx D_{KL}(\hat{S}||\hat{C}) \\
&= \frac{1}{2|\mathcal{E}|} \cdot \Big(\sum_{i=1}^{r^d} |H_i|_S \cdot \log |H_i|_S - \sum_{i=1}^{r^d} |H_i|_S \cdot \log \left(|H_i|_S + \Delta_i\right) \Big) \\
&\approx \frac{1}{2|\mathcal{E}|} \Big(\sum_{i=1}^{r^d} |H_i|_S \log |H_i|_S - \sum_{i=1}^{r^d} |H_i|_S \Big(\log |H_i|_S + \frac{\ln 2}{|H_i|_S} \Delta_i - \frac{\ln 2}{2(|H_i|_S)^2} \Delta_i^2 \Big) \Big) \\
&= \frac{1}{2|\mathcal{E}|} \cdot \sum_{i=1}^{r^d} \left(\frac{\ln 2}{2|H_i|_S} \Delta_i^2 - \ln 2 \cdot \Delta_i \right) \\
&= \frac{\ln 2}{4|\mathcal{E}|} \cdot \sum_{i=1}^{r^d} \frac{\Delta_i^2}{|H_i|_S},
\end{aligned}$$

which is the Chi-Square distance between $|H_i|_C$ and $|H_i|_S$. If we regard $|H_i|_S$ as constant, then we have: if Δ_i^2 are large, the information gain $D_{KL}(S||C)$ tends to be large. Formally, consider the samples of C as $\{x_v : v \in \mathcal{V}\}$ and the samples of S as $\{\frac{1}{|\mathcal{N}_v|} \sum_{v' \in \mathcal{N}_v} x_{v'} : v \in \mathcal{V}\}$ with counts $|\mathcal{N}_v|$ of node v, we have the expectation and variance of their distance as

$$\mathbb{E}\Big[|\mathcal{N}_v| \cdot x_v - \sum\nolimits_{v' \in \mathcal{N}_v} x_{v'} \Big] = 0,$$

$$\mathrm{Var}\Big(|\mathcal{N}_v| \cdot x_v - \sum\nolimits_{v' \in \mathcal{N}_v} x_{v'} \Big) = \mathbb{E}\Big[\big(|\mathcal{N}_v| \cdot x_v - \sum\nolimits_{v' \in \mathcal{N}_v} x_{v'} \big)^2 \Big] \geq 0.$$

For simplicity, for the distribution of the difference between the surrounding and the context, $|\mathcal{N}_v| \cdot x_v - \sum_{v' \in \mathcal{N}_v} x_{v'}$, we consider it as noises on the "expected" signal as the surrounding, $\frac{1}{|\mathcal{N}_v|} \cdot \sum_{v' \in \mathcal{N}_v} x_{v'}$, where the context x_v is the "observed" signal. Apparently the power of the noise on the samples is positively correlated with the difference between their PDFs, which means that a large $\mathrm{Var}\big(|\mathcal{N}_v| \cdot x_v - \sum_{v' \in \mathcal{N}_v} x_{v'} \big)$ would introduce a large difference between $|H_i|_S$ and $|H_i|_C$. Then, we obtain

$$\sum_{i=1}^{r^d} \frac{\Delta_i^2}{|H_i|_S} \sim \mathrm{Var}\Big(|\mathcal{N}_v| \cdot x_v - \sum\nolimits_{v' \in \mathcal{N}_v} x_{v'} \Big),$$

then it is easy to obtain

$$D_{KL}(S||C) \approx \frac{\ln 2}{4|\mathcal{E}|} \cdot \sum_{i=1}^{r^d} \frac{\Delta_i^2}{|H_i|_S} \sim \mathrm{Var}\Big(|\mathcal{N}_v| \cdot x_v - \sum\nolimits_{v' \in \mathcal{N}_v} x_{v'}\Big).$$

Recall the definition of λ_f, if x_{v_i} and x_{v_j} are independent, we have

$$\begin{aligned}
\lambda_f &= \frac{\Big|\Big| \sum_{v \in \mathcal{V}} \Big(\sum_{v' \in \mathcal{N}_v} (x_v - x_{v'}) \Big)^2 \Big|\Big|_1}{|\mathcal{E}| \cdot d} \\
&= \frac{\Big|\Big| \sum_{v \in \mathcal{V}} \Big(|\mathcal{N}_v| \cdot x_v - \sum_{v' \in \mathcal{N}_v} x_{v'} \Big)^2 \Big|\Big|_1}{|\mathcal{E}| \cdot d} \\
&= \frac{\Big|\Big| \mathrm{Var}\Big(|\mathcal{N}_v| \cdot x_v - \sum_{v' \in \mathcal{N}_v} x_{v'} \Big) \Big|\Big|_1}{|\mathcal{V}| \cdot |\mathcal{E}| \cdot d} \\
&\sim \mathrm{Var}\Big(|\mathcal{N}_v| \cdot x_v - \sum\nolimits_{v' \in \mathcal{N}_v} x_{v'} \Big).
\end{aligned}$$

Therefore, we obtain

$$D_{KL}(S||C) \approx \frac{\ln 2}{4|\mathcal{E}|} \cdot \sum_{i=1}^{r^d} \frac{\Delta_i^2}{|H_i|_S} \sim \mathrm{Var}\Big(|\mathcal{N}_v| \cdot x_v - \sum\nolimits_{v' \in \mathcal{N}_v} x_{v'} \Big) \sim \lambda_f.$$

If $\lambda_f = 0$, we have that all nodes have the same feature vector x_v. Obviously, $D_{KL}(\hat{S}||\hat{C}) = 0$. □

According to Theorem 4, a large λ_f means that a GNN model can obtain much information from graph data. Note that $D_{KL}(S||C)$ here is under the condition when using the *mean* aggregator. Others aggregators, e.g., *pooling* and *weight* could have different $D_{KL}(S||C)$ values, even if the feature smoothness λ_f is a constant.

4.1.2 Label Smoothness

After quantifying the information gain with λ_f, we next study how to measure the effectiveness of information gain. Consider the node classification task, where each node $v \in \mathcal{V}$ has a label y_v, we define $v_i \simeq v_j$ if $y_{v_i} = y_{v_j}$. The surrounding input can be decomposed into two parts based on the node labels as

$$\sum\nolimits_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k-1)} \breve{c}_{v_j}^{(k-1)} = \sum\nolimits_{v_j \in \mathcal{N}_{v_i}} \mathbb{I}(v_i \simeq v_j) a_{i,j}^{(k-1)} \breve{c}_{v_j}^{(k-1)} + \sum\nolimits_{v_j \in \mathcal{N}_{v_i}} (1 - \mathbb{I}(v_i \simeq v_j)) a_{i,j}^{(k-1)} \breve{c}_{v_j}^{(k-1)},$$

where $\mathbb{I}(\cdot)$ is an indicator function. The first term includes neighbors whose label y_{v_j} is the same as y_{v_i}, and the second term represents neighbors that have different labels. Assume that the classifier has good linearity, the label of the surrounding input is shifted to $\sum_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k-1)} \cdot y_{v_j}$ [63], where the label y_{v_j} is represented as a one-hot vector here. Note that in GNNs, even if the context and surrounding of v_i are combined, the label of v_i is still y_{v_i}. Thus, for the node classification task, it is reasonable to consider that neighbors with the same label contribute positive information and other neighbors contribute negative disturbance.

Definition 5 (*Label Smoothness*)**.** *To measure the quality of surrounding information, we define the label smoothness as*

$$\lambda_l = \sum\nolimits_{e_{v_i,v_j} \in \mathcal{E}} \big(1 - \mathbb{I}(v_i \simeq v_j)\big)/|\mathcal{E}|.$$

According to Definition 5, a larger λ_l implies that nodes with different labels tend to be connected together, in which case the surrounding contributes more negative disturbance for the task. In other words, a small λ_l means that a node can gain much positive information from its surrounding. To use λ_l to qualify the surrounding information, we require labeled data for the training. When some graphs do not have many labeled nodes, we may use a subset of labeled data to estimate λ_l, which is often sufficient for obtaining good results as we show for the BGP dataset used in our experiments.

In summary, we propose a context-surrounding framework, and introduce two smoothness metrics to estimate *how much information that the surrounding can provide* (i.e., larger λ_f means more information) and *how much information is useful* (i.e., smaller λ_l means more positive information) for a given task on a given graph.

4.2 Context-Surrounding Graph Neural Networks

In this section, we present a new GNN model, called **CS-GNN**, which utilizes the two smoothness metrics to improve the use of the information from the surrounding.

4.2.1 The Use of Smoothness for Context-Surrounding GNNs

The aggregator used in CS-GNN is *weighted sum* and the combination function is *concatenation*. To compute the coefficients for each of the K rounds, we use a multiplicative attention mechanism similar to [54]. We obtain $2|\mathcal{E}|$ attention coefficients by multiplying the leveraged representation vector of each neighbor of a node with

the node's context vector, and applying the softmax normalization. Formally, each coefficient $a_{i,j}^{(k)}$ in round k is defined as follows:

$$a_{i,j}^{(k)} = \frac{\exp\left(A(p_{v_i}^{(k)} \cdot q_{i,j}^{(k)})\right)}{\sum_{v_l \in \mathcal{N}_{v_i}} \exp\left(A(p_{v_i}^{(k)} \cdot q_{i,l}^{(k)})\right)}, \tag{2}$$

where $p_{v_i}^{(k)} = (W_p^{(k)} \cdot h_{v_i}^{(k)})^\top$, $q_{i,j}^{(k)} = p_{v_i}^{(k)} - W_q^{(k)} \cdot h_{v_j}^{(k)}$, $W_p^{(k)}$ and $W_q^{(k)}$ are two learnable matrices.

To improve the use of the surrounding information, we utilize feature and label smoothness as follows. First, we use λ_l to drop neighbors with negative information, i.e., we set $a_{i,j}^{(k)} = 0$ if $a_{i,j}^{(k)}$ is less than the value of the r-th ($r = \lceil 2|\mathcal{E}|\lambda_l \rceil$) smallest attention coefficient. As these neighbors contain noisy disturbance to the task, dropping them is helpful to retain a node's own features.

Second, as λ_f is used to estimate the quantity of information gain, we use it to set the dimension of $p_{v_i}^{(k)}$ as $\lceil d_k \cdot \sqrt{\lambda_f} \rceil$, which is obtained empirically to achieve good performance. Setting the appropriate dimension is important because a large dimension causes the attention mechanism to fluctuate while a small one limits its expressive power.

Third, we compute the attention coefficients differently from GAT [55]. GAT uses the leveraged representation vector $W^{(k)} \cdot h_{v_j}^{(k)}$ to compute the attention coefficients. In contrast, in equation (2) we use $q_{i,j}^{(k)}$, which is the difference of the context vector of node v_i and the leveraged representation vector of neighbor v_j. The definition of $q_{i,j}^{(k)}$ is inspired by the fact that a larger λ_f indicates that the features of a node and its neighbor are more dissimilar, meaning that the neighbor can contribute greater information gain. Thus, using $q_{i,j}^{(k)}$, we obtain a larger/smaller $a_{i,j}^{(k)}$ when the features of v_i and its neighbor v_j are more dissimilar/similar. For example, if the features of a node and its neighbors are very similar, then $q_{i,j}^{(k)}$ is small and hence $a_{i,j}^{(k)}$ is also small.

Using the attention coefficients, we perform K rounds of aggregations with the *weighted sum* aggregator to obtain the representation vectors for each node as

$$h_{v_i}^{(k)} = A\Big(W_l^{(k)} \cdot \big(h_{v_i}^{(k-1)} || \sum\nolimits_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k-1)} \cdot h_{v_j}^{(k-1)}\big)\Big),$$

where $W_l^{(k)}$ is a learnable parameter matrix to leverage feature vectors. Then, for a task such as node classification, we use a fully connected layer to obtain the final results $\hat{y}_{v_i} = A(W \cdot h_{v_i}^{(K)})$, where W is a learnable parameter matrix and $\hat{y}_{v_i}$ is the predicted classification result of node v_i.

4.2.2 Side Information on Graphs

Real-world graphs often contain *side information* such as attributes on both nodes and edges, local topology features and edge direction. We show that CS-GNN can be easily extended to include rich side information to improve performance. Generally speaking, side information can be divided into two types: context and surrounding. Usually, the side information attached on nodes belongs to the context and that on edges or neighbors belongs to the surrounding. To incorporate the side information into our CS-GNN model, we use the local topology features as an example.

We use a method inspired by GraphWave [14], which uses heat kernel in spectral graph wavelets to simulate heat diffusion characteristics as topology features. Specifically, we construct $|\mathcal{V}|$ subgraphs, $\mathbb{G} = \{G_{v_1}, G_{v_2}..., G_{v_{|\mathcal{V}|}}\}$, from a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where G_{v_i} is composed of v and its neighbors within K hops (usually K is small, $K = 2$ as default in our algorithm), as well as the connecting edges. For each $G_{v_i} \in \mathbb{G}$, the local topology feature vector t_{v_i} of node v_i is obtained by a method similar to GraphWave.

Since the topology feature vector t_{v_i} itself does not change during neighborhood aggregation, we do not merge it into the representation vector. In the attention mechanism, we regard t_{v_i} as a part of the context information by incorporating it into $p_{v_i}^{(k)} = (W_p^{(k)} \cdot (h_{v_i}^{(k)} || t_{v_i}))^\top$. And in the last fully connected layer, we use t_{v_i} to obtain the predicted class label $\hat{y}_{v_i} = A(W \cdot (h_{v_i}^{(K)} || t_{v_i}))$.

4.2.3 The Kullback–Leibler Divergence vs. Mutual Information

We use the Kullback–Leibler Divergence (KLD), instead of using Mutual Information (MI), to measure the information gain from the neighboring nodes in Section 4.1.1 because of the following reason. In an information diagram, mutual information $I(X;Y)$ can be seen as the overlap of two correlated variables X and Y, which is a symmetric measure. In contrast, $D_{KL}(X||Y)$ can be seen as the extra part brought by X to Y, which is a measure of the non-symmetric difference between two probability distributions. Considering the node classification task, the information contributed by neighbors and the information contributed to neighbors are different (i.e., non-symmetric). Thus, we use the KLD instead of MI.

We remark that, although some existing GNN works [56, 12] use MI in their models, their purposes are different from our work. MI can be written as $I(X,Y) = D_{KL}(P(X,Y)||P(X) \times P(Y))$, where $P(X,Y)$ is the joint distribution of X and Y, and $P(X)$, $P(Y)$ are marginal distributions of X and Y. From this perspective, we

can explain the mutual information of X and Y as the information loss when the joint distribution is used to approximate the marginal distributions. However, this is not our purpose in node classification.

4.2.4 Comparison with Existing GNN Models

The combination functions of existing GNNs are given in Table 2.1. The difference between additive combination and concatenation is that concatenation can retain a node's own feature. For the aggregation functions, different from GCN and GraphSAGE, GAT and CS-GNN improve the performance of a task on a given graph by an attention mechanism to learn the coefficients. However, CS-GNN differs from GAT in the following ways. First, the attention mechanism of CS-GNN follows multiplicative attention, while GAT follows additive attention. Second, CS-GNN's attention mechanism utilizes feature smoothness and label smoothness to improve the use of neighborhood information as discussed in Section 4.2.1. This is unique in CS-GNN and leads to significant performance improvements over existing GNNs (including GAT) for processing some challenging graphs (to be reported in Section 5.2.2). Third, CS-GNN uses side information such as local topology features to further utilize graph structure as discussed in Section 4.2.2.

CS-GNN enjoys the best from GCN, GraphSAGE and GAT as discussed above. We further analyze its advantages over GAT as they are most similar. GAT [55] learns the coefficients as

$$a_{i,j}^{(k)} = \frac{\exp\left(A(\boldsymbol{a}^\top [W^{(k)} \cdot h_{v_i}^{(k)} || W^{(k)} \cdot h_{v_j}^{(k)}])\right)}{\sum_{v_l \in \mathcal{N}_{v_i}} \exp\left(A(\boldsymbol{a}^\top [W^{(k)} \cdot h_{v_i}^{(k)} || W^{(k)} \cdot h_{v_l}^{(k)}])\right)}, \tag{3}$$

where the activation function $A(\cdot)$ is *LeakyReLU*, $\boldsymbol{a}$ is a parameterized vector and $W^{(k)}$ is a parameter matrix. However, this attention mechanism of GAT is not sensitive to graph topology. For example, two nodes with different local topology (e.g., 1 node with a single neighbor and the other with 100 neighbors) will not be differentiated by equation (3) if the features of their neighbors are the same. This is not desirable as graph data contain rich information in local topology features, which cannot be captured by GAT.

In contrast, CS-GNN is sensitive to graph topology. As discussed in Section 4.2.2, we use heat diffusion characteristics to extract topology features for each node based on its local topology. As proved theoretically in [14], nodes with similar topology have similar topology features. In contrast, existing random walk-based methods, e.g., [47], introduce uncertainties as nodes with similar topology only have similar topology features with certain probability. More importantly, GraphWave is also

capable of capturing complex topology features such as cliques and other substructures, in addition to other simpler side information such as node and edge attributes and edge directions.

CS-GNN is also sensitive to information gain. If we directly use traditional multiplicative attention on the node representation vector and its neighbor's representation vector, connected nodes with similar features tend to have large attention coefficients. As presented in equation (2), our attention mechanism is defined based on the difference of the context vector of a node and the leveraged representation vector of its neighbor, which gives a small attention coefficient for a neighbor with similar features.

Chapter 5

Experiments

5.1 Experimental Evaluation for PGE

We evaluated the performance of PGE using two benchmark applications, *node classification* and *link prediction*, which were also used in the evaluation of many existing graph embedding methods [22, 39, 20]. In addition, we also assessed the effects of various parameters on the performance of PGE.

Baseline Methods. We compared PGE with the representative works of the following three methods: *random walk based on skip-gram*, *graph convolutional networks*, and *neighbor aggregation based on weight matrices*.

- DeepWalk [43]: This work introduces the skip-gram model to learn node embeddings by capturing the relationships between nodes based on random walk paths. DeepWalk achieved significant improvements over its former works, especially for multi-labeled classification applications, and was thus selected for comparison.

- node2vec [21]: This method considers both graph homophily and structural equivalence. We compared PGE with node2vec as it is the representative work for graph embedding based on biased random walks.

- GCN [27]: This method is the seminal work that uses convolutional neural networks to learn node embedding.

- GraphSAGE [23]: GraphSAGE is the state-of-the-art graph embedding method and uses node property information in neighbor aggregation. It significantly improves the performance compared with former methods by learning the mapping function rather than embedding directly.

To ensure fair comparison, we used the optimal default parameters of the existing methods. For DeepWalk and node2vec, we used the same parameters to run the algorithms, with window size set to 10, walk length set to 80 and number of walks set to 10. Other parameters were set to their default values. For GCN, GraphSAGE and PGE, the learning rate was set to 0.01. For node classification, we set the epoch number to 100 (for GCN the early stop strategy was used), while for link prediction we set it to 1 (for PubMed we set it to 10 as the graph has a small number of nodes). The other parameters of GCN were set to their optimal default values. PGE also used the same default parameters as those of GraphSAGE such as the number of sampled layers and the number of neighbors.

Datasets. We used four real-world datasets in our experiments, including a citation network, a biological protein-protein interaction network and two social networks.

- **PubMed** [38] is a set of articles (i.e., nodes) related to diabetes from the PubMed database, and edges here represent the citation relationship. The node properties are TF/IDF-weighted word frequencies and node labels are the types of diabetes addressed in the articles.

- **PPI** [53] is composed of 24 protein-protein interaction graphs, where each graph represents a human tissue. Nodes here are proteins and edges are their interactions. The node properties include positional gene sets, motif gene sets and immunological signatures. The node labels are gene ontology sets. We used the processed version of [23].

- **BlogCatalog** [1] is a social network where users select categories for registration. Nodes are bloggers and edges are relationships between them (e.g., friends). Node properties contain user names, ids, blogs and blog categories. Node labels are user tags.

- **Reddit** [23] is an online discussion forum. The graph was constructed from Reddit posts. Nodes here are posts and they are connected if the same users commented on them. Property information includes the post title, comments and scores. Node labels represent the community. We used the sparse version processed in [23].

Table 5.1 shows some statistics of the datasets. To evaluate the performance of node classification of the algorithms on each dataset, the labels attached to nodes are treated as classes, whose number is shown in the last column. Note that each node in PPI and BlogCatalog may have multiple labels, while that in PubMed and Reddit has only a single label. The average degree (i.e., $|\mathcal{E}|/|\mathcal{V}|$) shows that the citation dataset

Table 5.1 Dataset statistics

Dataset	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	avg. degree	feature dim.	# of classes
PubMed	19,717	44,338	2.25	500	3
PPI	56,944	818,716	14.38	50	121
BlogCatalog	55,814	1,409,112	25.25	1,000	60
Reddit	232,965	11,606,919	49.82	602	41

PubMed is a sparse graph, while the other graphs have higher average degree. For undirected graphs, each edge is stored as two directed edges.

5.1.1 Node Classification

We first report the results for node classification. All nodes in a graph were divided into three types: training set, validation set and test set for evaluation. We used 70% for training, 10% for validation and 20% for test for all datasets except for PPI, which is composed of 24 subgraphs and we followed GraphSAGE [23] to use about 80% of the nodes (i.e., those in 22 subgraphs) for training and nodes in the remaining 2 subgraphs for validation and test. For the biases, we used the default values, $b_s = 1$ and $b_d = 1000$, for all the datasets. For the task of node classification, the embedding result (low-dimensional vectors) satisfies $\mathbf{z}_v \in \mathbb{R}^{d_l}$, where d_l is the number of classes as listed in Table 5.1. The index of the largest value in $\mathbf{z}_v$ is the classification result for single-class datasets. In case of multiple classes, the rounding function was utilized for processing $\mathbf{z}_v$ to obtain the classification results. We used F1-score [48], which is a popular metric for multi-label classification, to evaluate the performance of classification.

Table 5.2 reports the results, where the left table presents the F1-Micro values and the right table presents the F1-Macro values. PGE achieves higher F1-Micro and F1-Macro scores than all the other methods for all datasets, especially for PPI and BlogCatalog for which the performance improvements are significant. In general, the methods that use node property information (i.e., PGE, GraphSAGE and GCN) achieve higher scores than the methods that use the skip-gram model to capture the structure relationships (i.e., DeepWalk and node2vec). This is because richer property information is used by the former methods than the latter methods that use only the pure graph topology. Compared with GraphSAGE and GCN, PGE further improves the classification accuracy by introducing biases to differentiate neighbors for neighborhood aggregation, which validates our analysis on the importance of our

Table 5.2 Performance of node classification

F1-Micro (%) Alg. \ Datasets	PubMed	PPI	BlogCatalog	Reddit
DeepWalk	78.85	60.66	38.69	-
node2vec	78.53	61.98	37.79	-
GCN	84.61	-	-	-
GraphSAGE	88.08	63.41	47.22	94.93
PGE	**88.36**	**84.31**	**51.31**	**95.62**
F1-Macro (%) Alg. \ Datasets	PubMed	PPI	BlogCatalog	Reddit
DeepWalk	77.41	45.19	23.73	-
node2vec	77.08	48.57	22.94	-
GCN	84.27	-	-	-
GraphSAGE	87.87	51.85	30.65	92.30
PGE	**88.24**	**81.69**	**37.22**	**93.29**

biased strategy in Section 3.5. In the remainder of this subsection, we discuss in greater details the performance of the methods on each dataset.

To classify the article categories in PubMed, since the number of nodes in this graph is not large, we used the DBSCAN clustering method in Step 1, which produced $k = 4$ clusters. Note that the graph has a low average degree of only 2.25. Thus, differentiating the neighbors does not bring significant positive influence. Consequently, PGE's F1-scores are not significantly higher than those of GraphSAGE for this dataset.

To classify proteins' functions of PPI, since this graph is not very large, we also used DBSCAN for clustering, which produced $k = 39$ clusters. For this dataset, the improvement made by PGE over other methods is impressive, which could be explained by that neighbors in a protein-protein interaction graph play quite different roles and thus differentiating them may bring significantly benefits for node classification. In fact, although GraphSAGE also uses node property information, since GraphSAGE does not differentiate neighbors, it does not obtain significant improvement over DeepWalk and node2vec (which use structural information only). The small improvement made by GraphSAGE compared with the big improvement made by PGE demonstrates the effectiveness of our biased neighborhood sampling

Table 5.3 Performance of link prediction

MRR (%) Alg. \ Datasets	PubMed	PPI	BlogCatalog	Reddit
GraphSAGE	43.72	39.93	24.61	41.27
PGE (no edge info)	41.47	59.73	23.89	39.81
PGE	**70.77**	**89.21**	**72.97**	**56.59**

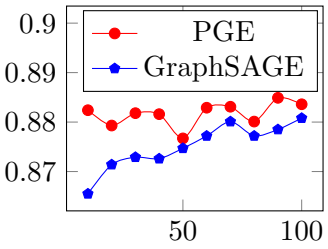
strategy. For GCN, since it does not consider multi-labeled classification, comparing it with the other methods is unfair and not meaningful for this dataset (also for BlogCatalog).

BlogCatalog has high feature dimensionality. The original BlogCatalog dataset regards the multi-hot vectors as the feature vectors (with $5,413$ dimensions). We used Truncate-SVD to obtain the low-dimensional feature vectors (with $1,000$ dimensions). Since the number of nodes is not large, we used DBSCAN for Step 1, which produced $k = 18$ clusters for this dataset. The improvement in the classification accuracy made by PGE is very significant compared with DeepWalk and node2vec, showing the importance of using property information for graph embedding. The improvement over GraphSAGE is also quite significant for this dataset, which is due to both neighbor differentiation and the use of edge direction.

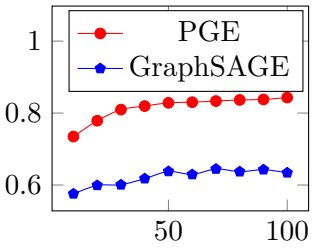
The Reddit graph is much larger than the other graphs, and thus we used K-Means (with $k = 40$) for clustering Reddit instead of using DBSCAN which is much slower. We do not report the results for DeepWalk and node2vec as their training processes did not finish in 10 hours while GraphSAGE and PGE finished in several minutes. We also do not report GCN since it needs to load the full graph matrix into each GPU and ran out of memory on our GPUs (each with 12GB memory). PGE's F1-scores are about 1% higher than those of GraphSAGE, which we believe is a significant improvement given that the accuracy of GraphSAGE is already very high (94.93% and 92.30%).

5.1.2 Link Prediction

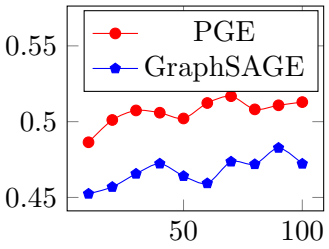
Next we evaluate the quality of graph embedding for link prediction. Given two nodes' embeddings $\mathbf{z}_v$ and $\mathbf{z}_{v'}$, the model should predict whether there is a potential edge existing between them. We used MRR (mean reciprocal rank) [46] to evaluate the performance of link prediction. Specifically, for a node v and $|Q|$ sets of nodes to



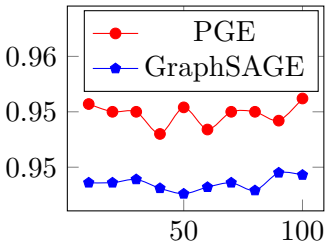
(a) F1-Micro PubMed



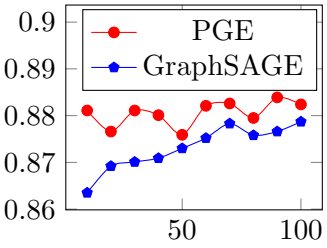
(b) F1-Micro PPI



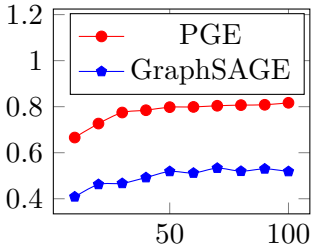
(c) F1-Micro BlogCatalog



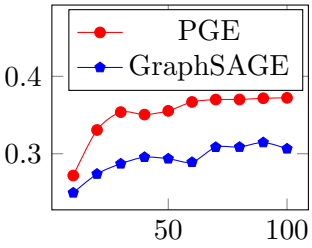
(d) F1-Micro Reddit



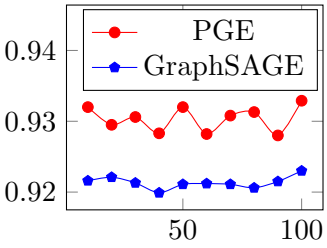
(e) F1-Macro PubMed



(f) F1-Macro PPI



(g) F1-Macro BlogCatalog



(h) F1-Macro Reddit

Figure 5.1: The effects of epoch number

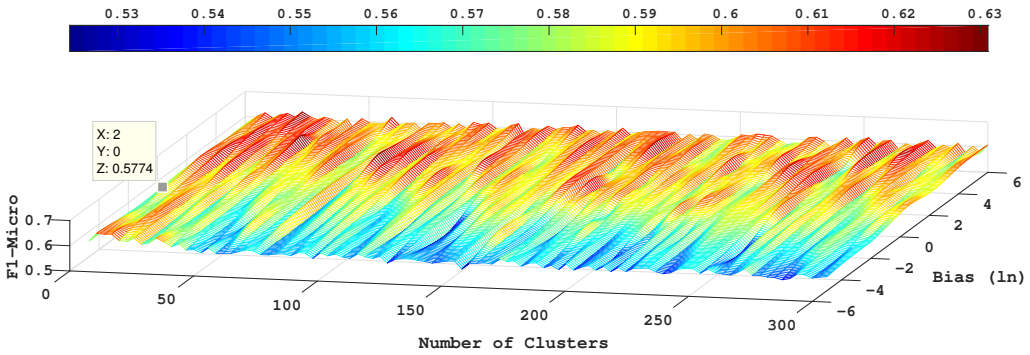


Figure 5.2: The effects of bias values and cluster number (best viewed as 2D color images)

be predicted, the MRR score can be calculated by the set of prediction queries/lists in Q with $\frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i}$, where $rank_i$ is the place of the first correct prediction. We compared PGE with GraphSAGE as we did not find the evaluation method for link prediction in DeepWalk, node2vec and GCN. For the sparse citation graph PubMed, we set the epoch number to 10 to avoid the data insufficiency problem. For other datasets, the epoch number was set to 1. As for the biases b_d and b_s and the clustering methods, they are the same as in the node classification experiment in Section 5.1.1.

Table 5.3 reports the MRR scores of PGE and GraphSAGE for the four datasets. We also created a variant of PGE by only considering bias (i.e., the edge information was not used). The results show that without considering the edge information, PGE records lower MRR scores than GraphSAGE except for PPI. However, when the edge information is incorporated, PGE significantly outperforms GraphSAGE in all cases and the MRR score of PGE is at least 37% higher than that of GraphSAGE. According to the MRR score definition, the correct prediction made by PGE is 1 to 3 positions ahead of that made by GraphSAGE. Compared with the improvements made by PGE for node classification, its improvements for link prediction are much more convincing, which can be explained as follows. Differentiating between neighboring nodes may not have a direct effect on predicting a link between two nodes; rather, the use of edge information by PGE makes a significant difference compared with GraphSAGE and the variant of PGE, as the latter two do not use edge information.

5.1.3 Parameter Sensitivity Tests

In this set of experiments, we evaluated the effects of the parameters in PGE on its performance.

Effects of the Epoch Number

To test the effects of the number of training epochs, we compared PGE with GraphSAGE by varying the epoch number from 10 to 100. We report the F1-Micro and F1-Macro scores for node classification on the four datasets in Figure 5.1. The results show that PGE and GraphSAGE have similar trends in F1-Micro and F1-Marco, although PGE always outperforms GraphSAGE. Note that the training time increases linearly with the epoch number, but the training time for 100 epochs is also only tens of seconds (for the small dataset PubMed) to less than 5 minutes (for the largest dataset Reddit).

Subfigures (a) and (e) present the F1-Micro and F1-Macro of PubMed. PGE performs better due to the biased neighbors sampling strategy when epoch number is low. With the increasing of epoch number, both methods have similar performance, with F1-Micro and F1-Macro around 0.88. This phenomenon is caused by that PubMed is too sparse to let the biased strategy take effects.

Subfigures (b) and (f) present the F1-Micro and F1-Macro of PPI. The performance on two methods are continuously increasing with the growth of epoch number. For GraphSAGE, the F1-Micro increases from 0.57 to 0.63, and the F1-Macro increases from 0.41 to 0.52. For PGE, its F1-Micro increases from 0.73 to 0.84, and F1-Macro increases from 0.67 to 0.82. Since nodes in PPI are protein molecules, the interaction between them is distinct. Therefore, to differentiate neighbors could bring significant positive effects.

Although BlogCatalog is a dense and multi-labeled graph, the number of nodes is small and its label matrix is very sparse. In Figure 5.1, small number of epochs makes two models under-fitting. With the growth of epoch number, PGE outperforms GraphSAGE gradually due to the biased strategy.

For the large-scale and dense graph Reddit, the performance of PGE is always better than that of GraphSAGE. With more details, the F1-Micro score of GraphSAGE decreases from 0.9486 to 0.9375 and the F1-Macro decreases from 0.922 to 0.908. As for PGE, the F1-Micro is around 0.955 and F1-Macro is around 0.929. With the growth of epoch number, GraphSAGE gradually becomes over-fitting, while our model could roughly maintain the performance.

Effects of Biases and Cluster Number

We also tested the effects of different bias values and the number of clusters. We ran PGE for 1,000 times for node classification on PPI, using different number of clusters k and different values of b_d (by fixing $b_s = 1$). We used K-Means for Step 1 since it is flexible to change the value k. The number of training epochs was set at 10 for each run of PGE. All the other parameters were set as their default values.

Figure 5.2 reports the results, where the X-axis shows the number of clusters k, the Y-axis indicates the logarithmic value (with the base e) of b_d, and the Z-axis is the F1-Micro score (F1-Macro score is similar and omitted). The results show that taking a larger bias b_d (i.e., $Y > 0$) can bring positive influence on the F1-score independent of the cluster number k, and the performance increases as a larger b_d is used. When b_d is less than 1, i.e., $b_d < b_s$, it does not improve the performance over uniform neighbor sampling (i.e., $b_d = b_s$ or $Y = 0$). This indicates that selecting a larger number of dissimilar neighbors (as a larger b_d means a higher probability of including dissimilar neighbors into $\mathcal{G}^s$) helps improve the quality of node embedding, which is consistent with our analysis in Section 3.5.

For the number of clusters k, as the average degree of the PPI graph is 14.38, when the cluster number is more than 50, the F1-score becomes fluctuating to k (i.e., the shape is like waves in Figure 5.2). This phenomenon is caused by the limitation of the clustering algorithm, since K-Means is sensitive to noises and a large k is more likely to be affected by noises. Note that when the cluster number is not large (less than 50), a small bias b_d (less than 1) may also improve the F1-score, which may be explained by the fact that there are homophily and structural equivalence features in the graph, while $b_d < 1$ indicates that nodes tend to select similar neighbors to aggregate. In general, however, a large b_d and a small cluster number k (close to the average degree) are more likely to improve the performance of the neighborhood aggregation method.

5.2 Experimental Evaluation for CS-GNN

We first compare CS-GNN with representative methods on the node classification task. Then we evaluate the effects of different feature smoothness and label smoothness on the performance of neural networks-based methods.

5.2.1 Baseline Methods, Datasets, and Settings

Baseline. We selected three types of methods for comparison: *topology-based methods, feature-based methods, and GNN methods*. For each type, some representatives were chosen. The topology-based representatives are **struc2vec** [47], **GraphWave** [14] and **Label Propagation** [66], which only utilize graph structure. struc2vec learns latent representations for the structural identity of nodes by random walk [43], where node degree is used as topology features. GraphWave is a graph signal processing method [40] that leverages heat wavelet diffusion patterns to represent each node and is capable of capturing complex topology features (e.g., loops and cliques). Label Propagation propagates labels from labeled nodes to unlabeled nodes. The feature-based methods are **Logistic Regression** and **Multilayer Perceptron** (**MLP**), which only use node features. The GNN representatives are **GCN**, **GraphSAGE** and **GAT**, which utilize both graph structure and node features.

Datasets. We used five real-world datasets: three citation networks (i.e., Citeseer, Cora [51] PubMed [38]), one computer co-purchasing network in Amazon [36], and one Border Gateway Protocol (BGP) Network [33]. The BGP network describes the Internet's inter-domain structure and only about 16% of the nodes have labels. Thus, we created two datasets: BGP (full), which is the original graph, and BGP (small), which was obtained by removing all unlabeled nodes and edges connected to them. The details (e.g., statistics and descriptions) of the datasets are given as follows.

Table 5.4 Dataset statistics

Dataset	Citeseer	Cora	PubMed	Amazon (computer)	BGP (small)	BGP (full)
$\lvert\mathcal{V}\rvert$	3,312	2,708	19,717	13,752	10,176	63,977
$\lvert\mathcal{E}\rvert$	4,715	5,429	44,327	245,861	206,799	349,606
Average degree	1.42	2.00	2.25	17.88	20.32	5.46
feature dim.	3,703	1,433	500	767	287	287
classes num.	6	7	3	10	7	7
λ_f (10^{-2})	2.7593	4.2564	0.9078	89.6716	7.4620	5.8970
λ_l	0.2554	0.1900	0.2455	0.2228	0.7131	$\approx$0.7131

Table 5.4 presents some statistics of the datasets used in our experiments, including the number of nodes $|\mathcal{V}|$, the number of edges $|\mathcal{E}|$, the average degree, the dimension of feature vectors, the number of classes, the feature smoothness λ_f and the label smoothness λ_l. The BGP (small) dataset was obtained from the original BGP dataset, i.e., BGP (full) in Table 5.4, by removing all unlabeled nodes and

edges connected to them. Since BGP (full) contains many unlabeled nodes, we used BGP (small)'s λ_l as an estimation. As we can see in Table 5.4, the three citation networks are quite sparse (with small average degree), while the other two networks are denser. According to the feature smoothness, Amazon (computer) has much larger λ_f than that of the others, which means that nodes with dissimilar features tend to be connected. As for the label smoothness λ_l, the BGP network has larger value than that of the others, meaning that connected nodes tend to belong to different classes. A description of each of these datasets is given as follows.

- Citeseer [51] is a citation network of Machine Learning papers that are divided into 6 classes: {Agents, AI, DB, IR, ML, HCI}. Nodes represent papers and edges model the citation relationships.
- Cora [51] is a citation network of Machine Learning papers that divided into 7 classes: {Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning, Theory}. Nodes represent papers and edges model the citation relationships.
- PubMed [38] 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 features are composed of TF/IDF-weighted word frequencies, and the node labels are the diabetes type addressed in the articles.
- Amazon Product [36] is a co-purchasing network derived from the Amazon platform. Nodes are computers and edges connect products that were bought together. Features are calculated from the product image, and labels are the categories of computers. We used the processed version by Alex Shchur in GitHub.
- Border Gateway Protocol (BGP) Network [33] describes the Internet's inter-domain structure, where nodes represent the autonomous systems and edges are the business relationships between nodes. The features contain basic properties, e.g., the location and topology information (e.g., transit degree), and labels means the types of autonomous systems. We used the dataset that was collected in 2018-12-31 and published in *Center for Applied Internet Data Analysis*.

Settings. We use F1-Micro score to measure the performance of each method for node classification. To avoid under-fitting, 70% nodes in each graph are used for

training, 10% for validation and 20% for testing. For each baseline method, we set their the parameters either as their default values or the same as in CS-GNN. For the GNNs and MLP, the number of hidden layers (rounds) was set as $K = 2$ to avoid over-smoothing. More detailed settings are given as follows.

For struc2vec and GraphWave, we used their default settings to obtain the node embeddings for all nodes since they are unsupervised. For struc2vec, we set their three optimization as "True". The embedding dimension of struc2vec is 128 and that of GraphWave is 100. Then based on those embeddings, we put them in a logistic model implemented by [43], where the embeddings of the training set were used to train the logistic model and the embeddings of the test set were used to obtain the final F1-Micro score. Label Propagation is a semi-supervised algorithm, where nodes in the training set were regarded as nodes with labels, and those nodes in the validation set and test set were regarded as unlabeled nodes.

The feature-based methods and GNN methods used the same settings: the batch size was set as 512; the learning rate was set as 0.01; the optimizer was Adam. Except GraphSAGE, which set the epoch number as 10, all the other methods were implemented with the early stop strategy with the patience number set as 100. Other parameters were slightly different for different datasets but still the same for all methods. Specifically, for Citeseer and Cora, the dropout was set as 0.2, and the weight decay was set as 0.01. And the hidden number was set as 8. For PubMed, the dropout was set as 0.3, but the weight decay was set as 0. The hidden number was 16. For Amazon (computer), BGP (small) and BGP (full), the dropout was set as 0.3, and the weight decay was set as 0. The hidden number was 32. As for GAT and CS-GNN, the attention dropout was set as the same as the dropout. And the dimension of topology features in CS-GNN was set as 64. Note that the residual technique introduced in GAT was used for the GNN methods. As for the activation function in CS-GNN, $ReLU$ was used for feature leveraging and ELU was used for the attention mechanism.

Note that GraphSAGE allows users to choose an aggregator. We tested four aggregators for GraphSAGE and report the best result for each dataset in our experiments below. Table 5.5 reports the F1-Micro scores of GCN and GraphSAGE with four different aggregators: GCN, mean, LSTM and max-pooling. The results show that for GraphSAGE, except on PubMed and BGP, the four aggregators achieve comparable performance. The results of GCN and GraphSAGE-GCN are worse than the others for PubMed and the BGP graphs because of the small information gain (i.e., small λ_f) of PubMed and the large negative disturbance (i.e., large λ_l) of BGP as reported in Table 5.7 and Section 5.2.2. As explained in Section 4.2.4, GCN uses additive combination merged with aggregation, where the features of each

Table 5.5 The F1-Micro scores of GraphSAGE with different aggregators

F1-Micro (%) / Dataset / Aggregators	Citeseer	Cora	PubMed	Amazon	BGP (small)	BGP (full)
GCN	71.27	80.92	80.31	91.17	51.26	54.46
GraphSAGE-GCN	68.57	**83.61**	81.76	88.14	49.64	48.54
GraphSAGE-mean	69.02	82.31	87.42	**90.78**	64.96	63.76
GraphSAGE-LSTM	69.17	82.50	87.08	90.09	**65.29**	**64.67**
GraphSAGE-pool (max)	**69.47**	82.87	**87.57**	87.39	65.06	64.24

node are aggregated with the features of its neighbors. As a result, GCN has poor performance for graphs with small λ_f and large λ_l, because it merges the context with the surrounding with negative information for a given task. This is further verified by the results of GraphSAGE using the other three aggregators, which still have comparable performance on all the datasets. In Section 5, we report the best F1-Micro score among these four aggregators for each dataset as the performance of GraphSAGE.

5.2.2 Performance Results of Node Classification

Smoothness. Table 5.6 reports the two smoothness values of each dataset. Amazon has a much larger λ_f value (i.e., 89.67×10^{-2}) than the rest, while PubMed has the smallest λ_f value. This implies that the feature vectors of most nodes in Amazon are dissimilar and conversely for PubMed. For label smoothness λ_l, BGP (small) has a fairly larger value (i.e., 0.71) than the other datasets, which means that 71% of connected nodes have different labels. Since BGP (full) contains many unlabeled nodes, we used BGP (small)'s λ_l as an estimation.

Table 5.6 Smoothness values

Smoothness value / Dataset / Metrics	Citeseer	Cora	PubMed	Amazon	BGP (small)	BGP (full)
Feature Smoothness λ_f (10^{-2})	2.76	4.26	0.91	89.67	7.46	5.90
Label Smoothness λ_l	0.26	0.19	0.25	0.22	0.71	$\approx$0.71

F1-Micro scores. Table 5.7 reports the F1-Micro scores of the different methods for the task of node classification. The F1-Micro scores are further divided into three groups. For the topology-based methods, Label Propagation has relatively

good performance for the citation networks and the co-purchasing Amazon network, which is explained as follows. Label Propagation is effective in community detection and these graphs contain many community structures, which can be inferred from their small λ_l values. This is because a small λ_l value means that many nodes have the same class label as their neighbors, while nodes that are connected together and in the same class tend to form a community. In contrast, for the BGP graph in which the role (class) of the nodes is mainly decided by topology features, struc2vec and GraphWave give better performance. GraphWave ran out of memory (512 GB) on the larger graphs as it is a spectrum-based method. For the feature-based methods, Logistic Regression and MLP have comparable performance on all the graphs.

For the GNN methods, GCN and GraphSAGE have comparable performance except on the PubMed and BGP graphs, and similar results are observed for GAT and CS-GNN. The main reason is that PubMed has a small λ_f, which means that a small amount of information gain is obtained from the surrounding, and BGP has large λ_l, meaning that most information obtained from the surrounding is negative disturbance. Under these two circumstances, using concatenation as the combination function allows GraphSAGE and CS-GNN to retain a node's own features. This is also why Logistic Regression and MLP also achieve good performance on PubMed and BGP because they only use the node features. However, for the other datasets, GAT and CS-GNN have considerably higher F1-Micro scores than all the other methods. Overall, CS-GNN is the only method that achieves competitive performance on all the datasets.

Table 5.7 Node classification results

F1-Micro(%) Alg. \ Dataset	Citeseer	Cora	PubMed	Amazon	BGP (small)	BGP (full)
struc2vec	30.98	41.34	47.60	39.86	48.40	49.66
GraphWave	28.12	31.66	OOM	37.33	50.26	OOM
Label Propagation	71.07	86.26	78.52	88.90	34.05	36.82
Logistic Regression	69.96	76.62	87.97	85.89	65.34	62.41
MLP	70.51	73.40	87.94	86.46	**67.08**	67.00
GCN	71.27	80.92	80.31	91.17	51.26	54.46
GraphSAGE	69.47	83.61	87.57	90.78	65.29	64.67
GAT	74.69	90.68	81.65	91.75	47.44	58.87
CS-GNN (w/o LTF)	73.58	90.38	89.42	92.48	66.20	**68.83**
CS-GNN	**75.71**	**91.26**	**89.53**	**92.77**	66.39	68.76

We further examine the effects of local topology features (LTF) on the performance. We report the results of CS-GNN without using LTF, denoted as **CS-GNN**

(w/o LTF) in Table 5.7. The results show that using LTF does not significantly improve the performance of CS-GNN. However, the results do reveal the effectiveness of the smoothness metrics in CS-GNN, because the difference between CS-GNN (w/o LTF) and GAT is mainly in the use of the smoothness metrics in CS-GNN's attention mechanism. As shown in Table 5.7, CS-GNN (w/o LTF) still achieves significant improvements over GAT on PubMed (by improving the gain of positive information) and BGP (by reducing negative noisy information from the neighborhood).

Improvements over non-GNN methods. We also evaluate whether GNNs are always better methods, in other words, whether graph information is always useful. Table 5.8 presents the improvements of existing GNNs (i.e., GCN, GraphSAGE, GAT) and CS-GNN over the topology-based and feature-based methods, respectively. The improvements (in %) are calculated based on the average F1-Micro scores of each group of methods. The results show that using the topology alone, even if the surrounding neighborhood is considered, is not sufficient. This is true even for the BGP graphs for which the classes of nodes are mainly determined by the graph topology. Compared with feature-based methods, GNN methods gain more information from the surrounding, which is converted into performance improvements. However, for graphs with small λ_f and large λ_l, existing GNN methods fail to obtain sufficient useful information or obtain too much negative noisy information, thus leading to even worse performance than purely feature-based methods. In contrast, CS-GNN utilizes smoothness to increase the gain of positive information and reduce negative noisy information for a given task, thus achieving good performance on all datasets.

Table 5.8 Improvements of GNNs over non-GNN methods

Improvement (%) / Dataset / Alg.	Citeseer	Cora	PubMed	Amazon	BGP (small)	BGP (full)
Existing GNNs vs. topology-based methods	65%	60%	32%	65%	23%	37%
CS-GNN vs. topology-based methods	74%	72%	42%	68%	50%	59%
Existing GNNs vs. feature-based methods	2%	13%	-5%	6%	-17%	-9%
CS-GNN vs. feature-based methods	8%	22%	2%	8%	0%	6 %

5.2.3 Smoothness Analysis

The results in Section 5.2.2 show that GNNs can achieve good performance by gaining surrounding information in graphs with large λ_f and small λ_l. However, the experiments were conducted on different graphs and there could be other factors than just the smoothness values of the graphs. Thus, in this experiment we aim to verify the effects of smoothness using one graph only.

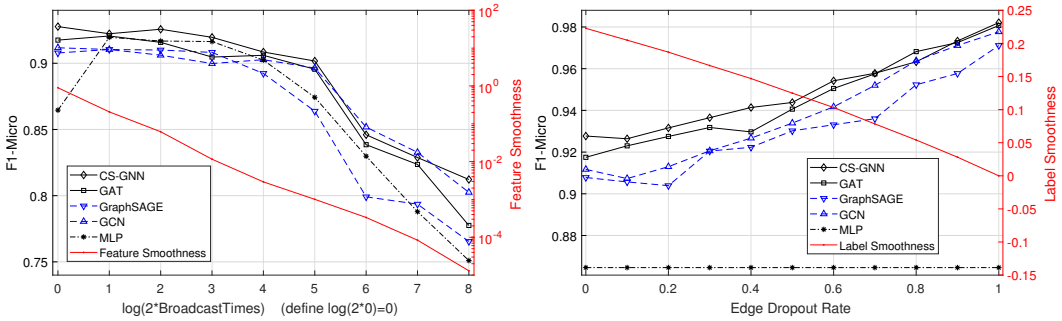


Figure 5.3: The effects of smoothness

To adjust λ_f in a graph, we broadcast the feature vector of each node to its neighbors in rounds. In each round, when a node receives feature vectors, it updates its feature vector as the mean of its current feature vector and those feature vectors received, and then broadcast the new feature vector to its neighbors. If we keep broadcasting iteratively, all node features converge to the same value due to over-smoothness. To adjust λ_l, we randomly drop a fraction of edges that connect two nodes with different labels. The removal of such edges decreases the value of λ_l and allows nodes to gain more positive information from their neighbors. We used the Amazon graph for the evaluation because the graph is dense and has large λ_f.

Figure 5.3 reports the F1-Micro scores of the neural-network-based methods (i.e., MLP and the GNNs). Figure 5.3 (left) shows that as we broadcast from 2^0 to 2^8 rounds, λ_f also decreases accordingly. As λ_f decreases, the performance of the GNN methods also worsens due to over-smoothness. However, the performance of MLP first improves significantly and then worsens, and becomes the poorest at the end. This is because the GNN methods can utilize the surrounding information by their design but MLP cannot. Thus, the broadcast of features makes it possible for MLP to first attain the surrounding information. But after many rounds of broadcast, the effect of over-smoothness becomes obvious and MLP's performance becomes poor. The results are also consistent with the fact that GNN models cannot be deep due to the over-smoothness effect. Figure 5.3 (right) shows that when λ_l decreases, the performance of the GNN methods improves accordingly. On the contrary, since MLP does not use surrounding information, dropping edges has no effect on its performance. In summary, Figure 5.3 further verifies that GNNs can achieve good performance on graphs with large λ_f and small λ_l, where they can obtain more positive information gain from the surrounding.

Chapter 6

Conclusion

First, we presented a representation learning framework, called PGE, for property graph embedding. The key idea of PGE is a three-step procedure to leverage both the topology and property information to obtain a better node embedding result. Our experimental results validated that, by incorporating the richer information contained in a property graph into the embedding procedure, PGE achieves better performance than existing graph embedding methods such as DeepWalk [43], node2vec [21], GCN [27] and GraphSAGE [23]. Second, we studied how to measure the quantity and quality of the information that GNNs can obtain from graph data. We then proposed CS-GNN to apply the smoothness measures to improve the use of graph information. We validated the usefulness of our method for measuring the smoothness values of a graph for a given task and that CS-GNN is able to gain more useful information to achieve improved performance over existing methods.

Appendix A

List of Publications

[1] **Yifan Hou**, Jian Zhang, James Cheng, Kaili Ma, Richard T. B. Ma, Hongzhi Chen, Ming-Chang Yang. Measuring and Improving the Use of Graph Information in Graph Neural Networks. *International Conference on Learning Representations*, 2020.

[2] **Yifan Hou**, Hongzhi Chen, Changji Li, James Cheng, Ming-Chang Yang. A Representation Learning Framework for Property Graphs. *International Conference on Knowledge Discovery & Data Mining*, pages 65-73, 2019.

[3] Hongzhi Chen, Changji Li, Juncheng Fang, Chenghuan Huang, James Cheng, Jian Zhang, **Yifan Hou**, Xiao Yan. Grasper: A High Performance Distributed System for OLAP on Property Graphs. *ACM Symposium on Cloud Computing*, pages 87-100, 2019.

[4] Hongzhi Chen, Xiaoxi Wang, Chenghuan Huang, Juncheng Fang, **Yifan Hou**, Changji Li, James Cheng. Large Scale Graph Mining with G-Miner. *International Conference on Management of Data*, pages 1881–1884, 2019.

Appendix B

Notations

The notations used in the paper and their descriptions are listed in Table B.1.

Table B.1 Notations and their descriptions

Notations	Descriptions
$\mathcal{G}$	A graph
$\mathcal{V}$	The set of nodes in a graph
v/v_i	Node v/v_i in $\mathcal{V}$
$\mathcal{E}$	The set of edges in a graph
e_{v_i,v_j}	An edge that connects nodes v_i and v_j
$\mathcal{X}$	The node feature space
x_v	The feature vector of node v
$y_v/\hat{y}_v$	The ground-truth / predicted class label of node v
$\mathcal{N}_v$	The set of neighbors of node v
$h_v/h_v^{(k)}$	The representation vector of node v (in round k)
$f(\cdot)$	A mapping function
W	A parameter matrix
$A(\cdot)$	An activation function
$a_{i,j}^{(k)}$	The coefficient of node v_j to node v_i (in round k)
$c_v^{(k)}$	The context vector of node v (in round k)
$\breve{c}_v^{(k)}$	The ground-truth context vector of node v (in round k)
$\breve{n}_v^{(k)}$	The noise on the context vector of node v (in round k)
$s_v^{(k)}$	The surrounding vector of node v (in round k)
d_k	The dimension of a representation vector (in round k)
$C^{(k)}$	Probability density function (PDF) estimated by the term $\breve{c}_v^{(k)}$ (in round k)
$S^{(k)}$	Probability density function (PDF) estimated by the term $\sum_{v_j \in \mathcal{N}_{v_i}} a_{i,j}^{(k)} \cdot \breve{c}_v^{(k)}$ (in round k)
$D_{KL}(S\|\|C)$	The Kullback–Leibler divergence between S and C
λ_f	Feature smoothness
λ_l	Label smoothness
$\|\|$	Vector concatenation
$\|\|\cdot\|\|_1$	Manhattan norm
$\mathbb{I}(\cdot)$	An indicator function
$\boldsymbol{a}$	Attention parameters (vector)
t_v	Topology feature of node v
G_{v_i}	A subgraph built based on node v_i
$\mathbb{G}$	The set of subgraphs G_{v_i}

Bibliography

[1] N. Agarwal, H. Liu, S. Murthy, A. Sen, and X. Wang. A social identity approach to identify familiar strangers in a social network. In *ICWSM*, 2009.

[2] A. Ahmed, N. Shervashidze, S. M. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *WWW*, pages 37–48, 2013.

[3] E. Bakshy, I. Rosenn, C. Marlow, and L. A. Adamic. The role of social networks in information diffusion. In *WWW*, pages 519–528, 2012.

[4] M. Barone and M. Coscia. Birds of A feather scam together: Trustworthiness homophily in A business network. *Social Networks*, 54:228–237, 2001.

[5] Y. Bengio, A. C. Courville, and P. Vincent. Representation learning: A review and new perspectives. *PAMI*, 35:1798–1828, 2013.

[6] S. Bhagat, G. Cormode, and S. Muthukrishnan. Node classification in social networks. In *Social network datanalytics*, pages 115–148. 2011.

[7] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34:18–42, 2017.

[8] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *CIKM*, pages 891–900, 2015.

[9] S. Cao, W. Lu, and Q. Xu. Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152, 2016.

[10] H. Chen, B. Perozzi, Y. Hu, and S. Skiena. HARP: hierarchical representation learning for networks. In *AAAI*, pages 2127–2134, 2018.

[11] J. Chen, J. Zhu, and L. Song. Stochastic training of graph convolutional networks with variance reduction. In *ICML*, pages 941–949, 2018.

[12] P. Chen, W. Liu, C. Hsieh, G. Chen, and S. Zhang. Utilizing edge features in graph neural networks via variational information maximization. *CoRR*, 2019.

[13] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, pages 3837–3845, 2016.

[14] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec. Learning structural node embeddings via diffusion wavelets. In *SIGKDD*, pages 1320–1329, 2018.

[15] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NeurIPS*, pages 2224–2232, 2015.

[16] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*, pages 226–231, 1996.

[17] S. Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, 2010.

[18] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur. Protein interface prediction using graph convolutional networks. In *NeurIPS*, pages 6530–6539, 2017.

[19] T. Gärtner, T. Horváth, and S. Wrobel. Graph kernels. In *Encyclopedia of Machine Learning and Data Mining*, pages 579–581. 2017.

[20] P. Goyal and E. Ferrara. Graph embedding techniques, applications, and performance: A survey. *KBS*, 151:78–94, 2018.

[21] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864, 2016.

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

[23] W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pages 1024–1034, 2017.

[24] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li. Rolx: Structural role extraction & mining in large graphs. In *SIGKDD*, pages 1231–1239, 2012.

[25] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

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

[27] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, 2016.

[28] T. N. Kipf and M. Welling. Variational graph auto-encoders. *CoRR*, 2016.

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

[30] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[31] W. Kurt. Kullback-leibler divergence explained, 2017.

[32] D. Liben-Nowell and J. M. Kleinberg. The link-prediction problem for social networks. *JASIST*, 58:1019–1031, 2007.

[33] M. J. Luckie, B. Huffaker, A. Dhamdhere, V. Giotsas, and kc claffy. AS relationships, customer cones, and validation. In *IMC*, pages 243–256, 2013.

[34] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[35] K. Marino, R. Salakhutdinov, and A. Gupta. The more you know: Using knowledge graphs for image classification. In *CVPR*, pages 20–28, 2017.

[36] J. J. McAuley, C. Targett, Q. Shi, and A. van den Hengel. Image-based recommendations on styles and substitutes. In *SIGIR*, pages 43–52, 2015.

[37] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[38] G. Namata, B. London, L. Getoor, B. Huang, and U. EDU. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, page 8, 2012.

[39] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104:11–33, 2016.

[40] A. Ortega, P. Frossard, J. Kovacevic, J. M. F. Moura, and P. Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, 2018.

[41] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *SIGKDD*, pages 1105–1114, 2016.

[42] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *WWW*, pages 1063–1072, 2018.

[43] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: online learning of social representations. In *SIGKDD*, pages 701–710, 2014.

[44] B. Perozzi, V. Kulkarni, and S. Skiena. Walklets: Multiscale graph embeddings for interpretable network classification. *CoRR*, 2016.

[45] L. R. Rabiner and B. Gold. Theory and application of digital signal processing. *Englewood Cliffs, NJ, Prentice-Hall, Inc.*, 1975.

[46] D. R. Radev, H. Qi, H. Wu, and W. Fan. Evaluating web-based question answering systems. In *LREC*, 2002.

[47] L. F. R. Ribeiro, P. H. P. Saverese, and D. R. Figueiredo. *struc2vec*: Learning node representations from structural identity. In *SIGKDD*, pages 385–394, 2017.

[48] Y. Sasaki. The truth of the f-measure. *Teach Tutor Mater*, 1:1–5, 2007.

[49] V. G. Satorras and J. B. Estrach. Few-shot learning with graph neural networks. In *ICLR*, 2018.

[50] M. S. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *ESWC*, pages 593–607, 2018.

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

[52] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, 2011.

[53] C. Stark, B. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. Biogrid: A general repository for interaction datasets. *Nucleic Acids Research*, 34:535–539, 2006.

[54] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[55] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *ICLR*, 2018.

[56] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm. Deep graph infomax. In *ICLR*, 2019.

[57] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *SIGKDD*, pages 1225–1234, 2016.

[58] Z. Wang, T. Chen, J. S. J. Ren, W. Yu, H. Cheng, and L. Lin. Deep reasoning with knowledge graph for social relationship understanding. In *IJCAI*, pages 1021–1028, 2018.

[59] B. Weisfeiler and A. A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

[60] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[61] J. Yang and J. Leskovec. Overlapping communities explain core-periphery organization of networks. *Proceedings of the IEEE*, 102:1892–1902, 2014.

[62] L. Yao, C. Mao, and Y. Luo. Graph convolutional networks for text classification. In *AAAI*, pages 7370–7377, 2019.

[63] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018.

[64] D. Zhou and B. Schölkopf. A regularization framework for learning from graph data. In *ICML workshop*, volume 15, pages 67–68, 2004.

[65] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods and applications. *CoRR*, 2018.

[66] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.