Noname manuscript No.
(will be inserted by the editor)

OpenWGL: Open-World Graph Learning for Unseen Class Node Classification

Man Wu · Shirui Pan · Xingquan Zhu

Received: 21 Jan 2021 / Revised: 07 May 2021 / Accepted: 19 Jun 2021

Abstract Graph learning, such as node classification, is typically carried out in a *closed-world* setting. A number of nodes are labeled, and the learning goal is to correctly classify remaining (unlabeled) nodes into classes, represented by the labeled nodes. In reality, due to limited labeling capability or dynamic evolving nature of networks, some nodes in the networks may not belong to any existing/seen classes, and therefore cannot be correctly classified by closed-world learning algorithms. In this paper, we propose a new *open-world* graph learning paradigm, where the learning goal is to correctly classy nodes belonging to labeled classes into correct categories, and also classify nodes not belonging to labeled classes to an unseen class. Open-world graph learning has three major challenges: (1) graphs do not have features to represent nodes for learning; (2) unseen class nodes do not have labels, and may exist in an arbitrary form different from labeled classes; and (3) graph learning should differentiate whether a node belong to an existing/seen class or an unseen class. To tackle the challenges, we propose an uncertain node representation learning principle to use multiple versions of node feature representation to test a classifier's response on a node, through which we can differentiate whether a node belongs to the unseen class. Technical wise, we propose constrained variational graph autoencoder, using label loss and class uncertainty loss constraints, to

M. Wu
Dept. of Computer & Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA
E-mail: mwu2019@fau.edu

S. Pan (✉)
Department of Data Science and AI,Faculty of IT, Monash University, Clayton, Melbourne, VIC 3800 Australia
E-mail: shirui.pan@monash.edu

X. Zhu (✉)
Dept. of Computer & Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA
E-mail: xzhu3@fau.edu

ensure that node representation learning is sensitive to the unseen class. As a result, node embedding features are denoted by distributions, instead of deterministic feature vectors. In order to test the certainty of a node belonging to seen classes, a sampling process is proposed to generate multiple versions of feature vectors to represent each node, using automatic thresholding to reject nodes not belonging to seen classes as unseen class nodes. Experiments, using graph convolutional networks and graph attention networks on four real-world networks, demonstrate the algorithm performance. Case studies and ablation analysis also show the advantage of the uncertain representation learning and automatic threshold selection for open-world graph learning.

Keywords Graph neural network, uncertain node representation learning, open-world learning, node classification

1 Introduction

Networks/Graphs are convenient tools to model interactions and interdependencies between large-scale data. Graph learning, such as node classification[1], attempts to categorize nodes of graphs into several groups. Such learning tasks are fundamental, but challenging, and have received continuous attention in the research field. Many research efforts have been made to develop reliable and efficient algorithms for different types of node classification tasks. However, existing methods mainly carry out the learning in a "closed-world" setting, where classes in the test set must be consistent to the classes used in the training set. In other words, nodes in the test data must belong to classes already seen in the training set. As a result, when a new/unseen class node appears in the test set, classifiers cannot detect the new/unseen class and will erroneously classify the node to seen/learned classes in the training data.

In reality, data collection and labeling may be continuously evolving. New trends emerge constantly and a model that cannot detect these new/unseen trends can hardly work well in a dynamic environment. This problem/phenomenon is referred to as the *open-world classification* or *open classification* problem [1]. The new "open-world" learning (OWL) [2–4] paradigm is to not only recognize objects belonging to the classes already seen/learned before, but also detect new class samples which are previously unseen.

Several approaches, such as one-class SVM [5], can be adjusted to address open-world learning by treating all seen classes as the positive class, but they cannot differentiate instances in seen classes and often have poor performance to find unseen class, because no negative data is used. Alternatively, a similar problem called covariate shift [6] has also been studied in social media text classification, where covariate shift means that training data are not fully representative of the test data. To address the problem, a Center-Based Similarity (CBS) space learning method [6] firstly computes a center for each class and

[1] Note that node classification in this article refers to single-label classification.

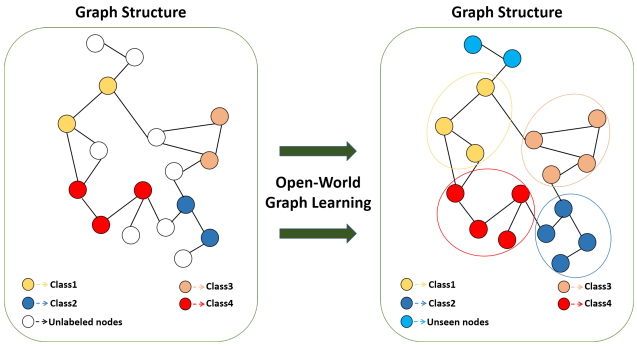


Fig. 1: An example of open-world learning for network node classification. Nodes are either labeled or unlabeled. Given a graph with some labeled nodes and unlabeled nodes (left panel), open-world graph learning aims to learn a classifier to classify unlabeled nodes belonging to seen classes into its own class, and also detects unlabeled nodes not belonging to any seen class as unseen class nodes (denoted by green colored nodes in the right panel).

converts a document to a vector of similarities to the center. The transformed data is then used to build a binary classifier for each class.

To date, open-world learning has already attracted many interests from Natural Language Processing (NLP) [1] [2] and computer vision fields [7] [8] [9]. In NLP, Shu et al. [1] proposed the solution to open-world learning by setting thresholds before the sigmoid function to reject data from the unseen class. In computer vision, Scheirer et al. [7] studied the problem of recognizing unseen images that are not in the training data by reducing the half-space of a binary SVM classifier with a positive region. However, to the best of our knowledge, the open-world classification problem has not been previously investigated in graph structure data and graph learning tasks.

Given a graph consisting of seen class and unseen class nodes, the objective of open-world graph learning is to build a classifier to classify nodes belonging to seen classes into correct categories, and also detect nodes not belonging to any seen class as unseen class. An example of open-world learning for graph node classification is illustrated in Fig. 1.

Currently, existing solutions to open-world learning are mainly focused on documents or images, and cannot be directly applied to graph structured data and graph learning tasks because they cannot model graph structural information, which is the core of node classification.

The challenge of graph learning is that graphs have node content and structure information where nodes are connected with edges representing their relations. Furthermore, existing solutions to node classification task are built on the closed-world assumption, in which the classes appeared in the testing data must have appeared in training. For example, the basic idea of graph convolutional networks (GCNs) is to develop a convolutional layer to exploit

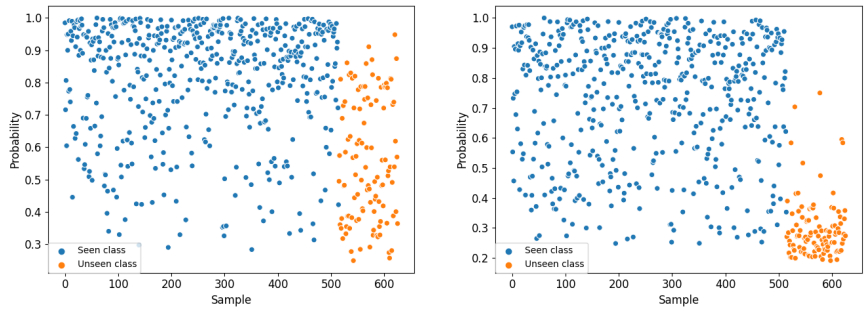


Fig. 2: A visualization of classification probability on seen (blue) and unseen (orange) class test instances for Cora dataset. The $x-$axis denotes the index of test instances (first 500 instances belong to seen classes and the last 100 instances belong to unseen class). The $y-$axis denotes the maximum probability output of each instance through the softmax classifier. (a) denotes the classification probabilities using only label loss, and (b) denotes the classification probabilities combining both label loss and class uncertainty loss.

the graph structure information and use a classification loss function to guide the classification task. However, they directly use softmax as the final output layer, which does not have the rejection capability to unseen class nodes because the prediction probability of each class is normalized across all training/seen classes. In addition, in representation learning level, most existing graph learning methods employ feature engineering or deep learning to extract feature vectors. However, these models can only generate deterministic mappings to capture latent features of nodes. A major limitation of them is their inability to represent uncertainty caused by incomplete or finite available data.

In this paper, we propose to study open-world learning for graph data. Considering the complicated graph data structure and the node classification task, we summarize the main challenges as follows,

- *Challenge 1*: How to design an end to end framework for open-world graph learning in graphs where the unseen class has no labeled samples, and may exist in an arbitrary form different from seen classes. Existing graph neural networks (GNNs) are typical built based on closed-world assumption and cannot detect unseen class.
- *Challenge 2*: How can we model the uncertainty of node representations and promote robustness in graphs. Many existing GNN-based approaches only generate deterministic mappings to capture latent features of nodes.

To overcome the above challenges, we propose a novel open-world graph leaning paradigm (OpenWGL) for the node classification task. For *Challenge 1*, we employ two loss constraints (a label loss and a class uncertainty loss) to

ensure that the node representation learning is sensitive to unseen class and assist in our model to differentiate whether a node belongs to an existing/seen class or the unseen class. We visualize a testing dataset in our experiment in Fig. 2, which can illustrate the effectiveness of our method. In Fig. 2(a), we only use the label loss (the cross-entropy loss), which has a good performance on existing/seen class nodes, but unseen class nodes cannot be differentiated and will be classified to seen classes randomly. In Fig. 2(b), we introduce a class uncertainty loss constraint, which can reduce the probability of unseen class nodes being classified as the seen class, and therefore help detect unseen class nodes without reducing the classification performance for nodes in seen classes. For *Challenge 2*, instead of learning a deterministic node feature vector, we utilize a graph variational autoencoder module to learn a latent distribution to represent each node. During the classification phase, a novel sampling process is used to generate multiple versions of feature vectors to test the certainty of a node belonging to seen classes, and automatically determine a threshold to reject nodes not belonging to seen classes as unseen class nodes.

Our contributions can be summarized as follows:

- We formulate a new *open-world learning* problem for graph data, and present a novel deep learning model OpenWGL as a solution.
- We propose an uncertain node representation learning approach, by using label loss and class uncertainty loss to constrain variational graph autoencoder to learn node representation sensitive to unseen class.
- We propose to use sampling process to test the certainty of a node belonging to seen classes, and automatically determine a threshold to reject nodes not belonging to seen classes as unseen class nodes.
- Experiments on benchmark graph datasets demonstrate that our approach outperforms the baseline methods.

2 Related work

This work is closely related to open-world learning, emerging class and outlier detection, and graph neural networks, which are briefly reviewed below.

2.1 Open-World Learning

Open-World Learning aims to recognize the classes the learner has seen/learned before and also detect a new class it has never seen before. There are some early explorations of open-world learning. Scholkopf et al. [5] employ the one-class SVM as the classifier, which shows poor performance since no negative data is used. Fei and Liu [6] propose a Center-Based Similarity (CBS) space learning method, which first computes a center for each class and converts each document to a vector of similarities to the center. The transformed data is then used to build a binary classifier for each class. Fei et al. [3] then extend

their work by adding the capability of incrementally or cumulatively learning new classes.

Recently, open-world learning has been studied in Natural Language Processing [1] [2] and Computer Vision (where it is called open-set recognition) [7] [8] [9]. In NLP, Shu et al. [1] propose the deep learning solution to open-world learning by setting thresholds before the sigmoid function to reject unseen classes. Xu et al. [2] propose a new open-world learning model based on meta-learning, which allows new classes to be added or deleted with no need for model re-training. In computer vision, Scheirer et al. [7] study the problem of recognizing unseen images that are not in the training data by reducing the half-space of a binary SVM classifier with a positive region. In [8] and [9], Scheirer et al. utilize the probability threshold to detect new classes, while their models are weak because of lacking prior knowledge.

Most existing open-world learning approaches are primarily focused on NLP and CV domains, and cannot model graph structural data. In our research [10], we proposed to advance the open-world learning principle to graph data, and designed a graph learning framework to classify network nodes in an open-world setting.

2.2 Emerging Class and Outlier Detection

Our research is also related to emerging/new class detection in supervised learning, such as stream data mining [11, 12] and multi-instance learning [13], and outlier detection [14].

In supervised learning, instances are assumed to belong to at least one of the predefined classes, and a classifier is trained to learn discriminative patterns to separate samples into known classes. In reality, all data patterns may not be known when the data is collected, or new classes may emerge over time. When a class is unknown or unavailable at the time of training a classifier, in the test stage, an ideal classifier is expected to be able to detect the emerging/new class [15]. A common solution of detecting new class samples is to use a decision threshold to give a confidence score [16–18], including multilayer neural network [19] to increase the threshold, and samples with low confidence below threshold are recognized as the new class. Unfortunately, as we have shown in Fig. 2, simply increasing the threshold will make existing class samples being misclassified.

Outlier detection, on the other hand, aims to detect data instances which abnormally deviate from the underlying data [20]. Akoglu et al. [21] provide a comprehensive overview of graph-based techniques for anomaly, event, and fraud detection, as well as their use for post-analysis and sense-making in explaining the detected abnormalities. Some distance-based outlier detection methods such as One-class SVM have been proposed, in which the normal data domain is obtained by finding a hyper-sphere enclosing the normal data samples [5] [22]. For all methods, there is a trade-off between the number of true outliers and false outliers (samples being detected as outliers but come from

the same distribution as the training data) [15]. A recent proposed method called StrOUD utilizes transduction and statistical tests to measure the fitness of cluster structures [23]. A recent method [14] proposes to detect outliers from data stream, but new class detection by outliers is not addressed.

In summary, our research not only advances the emerging (new) class detection to networked data settings, but also proposes a new way of automatically determine threshold for open-world learning.

2.3 Graph Neural Networks

Graph neural networks (GNNs), introduced in [24] and [25] as a generalization of recursive neural networks to directly deal with a more general class of graphs, *e.g.* cyclic, directed and undirected graphs, are a powerful tool for machine learning on graphs. GNNs have attracted attention all around the world, which are designed to use deep learning architectures on graph-structured data [26] [27] [28]. Many solutions are proposed to generalize well-established neural network models that work on regular grid structure to deal with graphs with arbitrary structures [29] [30] [31].

Among these methods, the most classic model is graph convolutional network (GCN), which is a deep convolutional learning paradigm for graph-structured data integrating local node features and graph topology structure in convolutional layers [32]. GraphSAGE [33] is a variant of GCN which designs different aggregation methods for feature extraction. GAT [34] improves GCN by leveraging attention mechanism to aggregate features from the neighbors of a node with discrimination. Although GCNs have shown great performance in graph-structured data for semi-supervised learning tasks such as node classification, the variational graph autoencoder (VGAE) [35] extends it to unsupervised scenarios. Specifically, VGAE integrates GCN into the variational encoder framework [36] by using a graph convolutional encoder and a simple inner product decoder.

For existing GCN-based graph learning models, they are built on the closed-world assumption, in which the classes appeared in the test data must have shown in training. In this paper, We employ two loss constrains to ensure that the node representation learning is sensitive to unseen class and assist in our model differentiating whether a node may belong to an existing/seen class or an unseen class.

To the best of our knowledge, the open-world learning problem has not been previously investigated in graph structure data and graph learning tasks. We are the first to study the *open-world graph learning* and propose an novel uncertain node representation learning approach, based on a variant of GCN (*i.e.*, variational graph autoencoder networks) to differentiate whether a node belongs to an existing (seen) class or an unseen class.

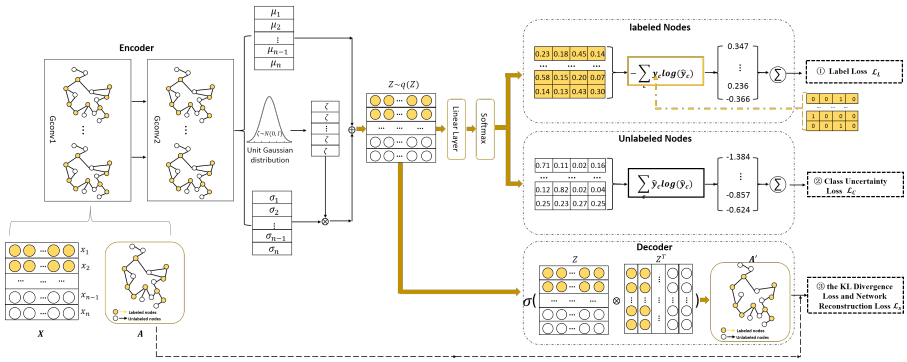


Fig. 3: The overall architecture of the proposed Open-World Graph Learning (OpenWGL) model for unseen class node classification. The input consists of a graph with labeled and unlabeled nodes. The learning objective of OpenWGL, defined by Eq. (12), is constrained by ① label loss ($\mathcal{L}_L$) defined by Eq. (13), ② class uncertainty loss ($\mathcal{L}_C$) defined by Eq. (14), and ③ the KL divergence loss and network reconstruction loss ($\mathcal{L}_S$) defined by Eq. (11). As a result, OpenWGL can learn uncertain node representation sensitive to the class labels and unseen class. More specifically, OpenWGL first uses uncertain node representation learning to generate a latent distribution of each node, which consists of a graph encoder model and a graph decoder model. After that, a sampling process is employed to the latent distribution to learn solutions to an objective function which combines the three loss constraints (structure loss, label loss, and class uncertainty loss). More details are given in Section 4.

3 Problem Definition and Overall Framework

This section defines the problem to be addressed in our paper and then presents our overall framework for the problem.

3.1 Problem Statement

Node Classification on Graphs: In this paper, we focus on node classification on graphs. A graph is represented as $G = (V, E, X, Y)$, where $V = \{v_i\}_{i=1,\cdots,N}$ is a vertex set representing nodes in a graph, and $e_{i,j} = (v_i, v_j) \in E$ is an edge indicating the relationship between two nodes. The topological structure of a graph G can be represented by an adjacency matrix A, where $A_{i,j} = 1$ if $(v_i, v_j) \in E$; otherwise $A_{i,j} = 0$. $x_i \in X$ indicates content features associated with each node v_i. $Y \in \mathbb{R}^{N \times C}$ is a label matrix of G, where N is the number of nodes in G and C is the number of node categories (classes) already known/seen. If a node $v_i \in V$ is associated with label l , $Y^l_{(i)} = 1$; otherwise, $Y^l_{(i)} = 0$.

Open-World Graph Learning: Given a graph $G = (V, E, X, Y)$, $X = X_{train} \bigcup X_{test}$, where X_{train} denotes training data (labeled nodes) and X_{test} denotes testing nodes (unlabeled nodes). Assume $X_{test} = S \bigcup U$, where S are the set of nodes belonging to seen classes already appeared in X_{train} and U are the set of nodes not belonging to any seen class (*i.e.* unseen class nodes). Open-World Learning on Graphs **aims** to learn a $(C+1)$-class classifier model, $f(X_{test}) \mapsto Y$, $(Y \in \{1, \cdots, C, rejection\})$ to classify each test node S to one of the training/seen classes in Y and reject U to indicate that it does not belong to any training/seen class (*i.e.*, it belongs to the unseen class).

3.2 Overall Framework

In order to learn a classifier for open-world graph learning, we propose an uncertain node representation learning approach called Constrained Variational Graph Autoencoder network to classify each seen node to its accurate category and reject the unseen nodes. Our framework for open-world graph learning, as shown in Fig. 3, mainly consists of following two components:

- **Node Uncertainty Representation Learning**. Most GCN models generate deterministic mappings to capture latent features of nodes. A major limitation of these models is their inability to represent uncertainty caused by incomplete or finite available data. In order to learn a better representation of each node, we employ a Variational Graph Autoencoder Network to obtain a latent distribution of each node, which enables to represent uncertainty and promote robustness.
- **Open-World Classifier Learning**. In order to classify seen class nodes to their own groups and detect unseen class nodes, we introduce two constraints, label loss and class uncertainty loss, to differentiate whether a node belongs to an existing class or an unseen class.

Open-World Classification & Rejection. To perform inference during the testing phase (i.e., perform classification or rejection of an example), we propose a novel sampling process to generate multiple versions of feature vectors to test the certainty of a node belonging to seen classes and automatically determine a threshold to reject nodes not belonging to seen classes as unseen nodes. Our inference framework is given in Fig. 4 with detailed discussion given in Section IV. C.

4 Methodology

4.1 Node Uncertainty Representation Learning

In order to encode latent feature information of each node and obtain an effective representation of uncertainty, we employ Variational Graph Autoencoder network (VGAE) to generate a latent distribution based on extracted node

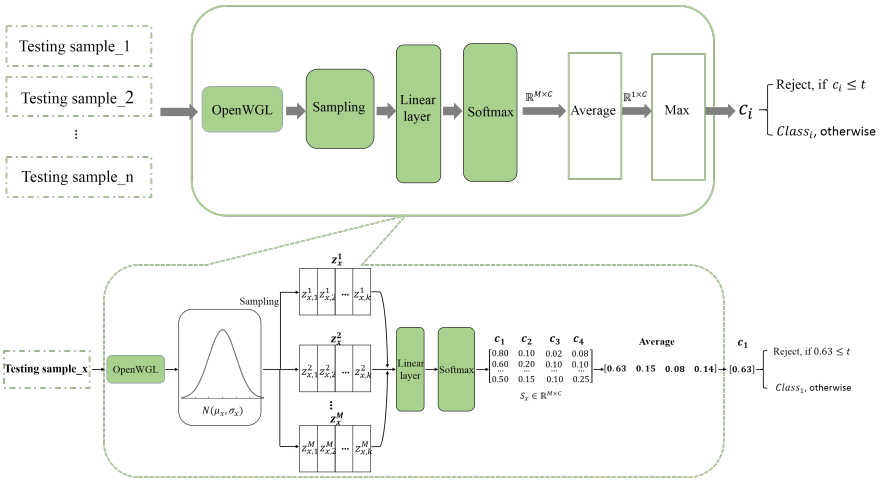


Fig. 4: The classification and rejection process (assuming seen class set has 4 classes). For nodes in the testing set, node uncertainty representation learning generates M different versions of feature vectors for each node by a sampling process. The M different representations are fed into a softmax layer to obtain M probability outputs S_i. The probabilities of each class are averaged to obtain a vector $s_{i,a}$, and the largest average is denoted by $max(s_{i,a})$. Finally, Eq.(15) is used to decide whether a node belongs to the seen or unseen classes.

features. This allows our method to leverage uncertainty for robust representation learning.

Graph Encoder Model: Given a graph $G = (X, A)$, in order to represent both node content X and graph structure A in a unified framework, our approach firstly utilizes a two-layer GNNs to map the feature matrix. Several classical GNNs, such as GCN [32] and GAT [34], are tested as the backbone of the two-layer GNNs. Given the input feature matrix X and adjacency matrix A, the first GCN layer generates a lower-dimensional feature matrix, which is defined as follows:

$$Z^{(1)} = \mathsf{GNN}(X, A) \tag{1}$$

For the second layer GNN model, instead of generating a deterministic representation, we assume that the output Z is continuous and follows a multivariate Gaussian distribution. Hence, we follow an inference model proposed by [35]:

$$q(Z|X, A) = \prod_{i=1}^{N} q(\boldsymbol{z}_i|X, A), \tag{2}$$

$$q(\boldsymbol{z}_i|X, A) = \mathcal{N}(\boldsymbol{z}_i|\boldsymbol{\mu}_i, diag(\boldsymbol{\sigma}_i^2)) \tag{3}$$

Here, $\boldsymbol{\mu} = \mathsf{GNN}_{\mu}(X, A)$ is the matrix of mean vectors μ_i; $\boldsymbol{\sigma}$ is the standard variance matrix of the distribution, $\log\boldsymbol{\sigma} = \mathsf{GNN}_{\sigma}(X, A)$. Then we can calcu-

late Z using a parameterization trick:

$$Z = \boldsymbol{\mu} + \boldsymbol{\sigma} \cdot \zeta, \zeta \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{4}$$

where $\mathbf{0}$ is a vector of zeros and $\boldsymbol{I}$ is the identity matrix. By making use of the latent variable Z, our model is able to capture complex noisy patterns in the data.

For each layer of GNNs, the calculation process is as follows:

1) Graph Convolutional Networks. The lth GCN layer inputs a feature matrix $\mathbf{Z}^l \in \mathbb{R}^{c \times d_{(l)}}$ and outputs a higher-order feature matrix $\mathbf{Z}^{l+1} \in \mathbb{R}^{c \times d_{(l+1)}}$, which can be written as a non-linear function:

$$\begin{aligned} \mathbf{Z}^0 &= \mathbf{X}, \\ \mathbf{Z}^{l+1} &= \sigma(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{Z}^l\mathbf{W}^l) \end{aligned} \tag{5}$$

where the degree matrix $\mathbf{D}_{ij} = \sum_j \mathbf{A}_{ij}$ is a diagonal matrix, $\mathbf{W}^l \in \mathbb{R}^{d_{(l)} \times d_{(l+1)}}$ is the transformation matrix for the lth layer and $\sigma(\cdot)$ is a non-linear activation function, which is acted by ReLU in our experiments.

2) Graph Attention Networks. The lth GAT layer with single head inputs a feature matrix $\mathbf{Z}^l \in \mathbb{R}^{c \times d_{(l)}}$ and apply a shared linear transformation, parametrized by a weight matrix $\mathbf{W}^l \in \mathbb{R}^{d_{(l+1)} \times d_{(l)}}$, to each node. Then a shared attention mechanism is leveraged to compute attention coefficients of the pairs of connected nodes:

$$\begin{aligned} \mathbf{e}_{ij}^l &= a^l(\mathbf{W}^l\mathbf{Z}_i^l, \mathbf{W}^l\mathbf{Z}_j^l) \\ &= LeakyRelu(\mathbf{a}^l[\mathbf{W}^l\mathbf{Z}_i^l || \mathbf{W}^l\mathbf{Z}_j^l]^{\mathrm{T}}) \end{aligned} \tag{6}$$

where $a^l(\cdot,\cdot) : \mathbb{R}^{d_{(l+1)}} \times \mathbb{R}^{d_{(l+1)}} \to \mathbb{R}$ is a single-layer feed forward neural network which is parametrized by a weight vector $\mathbf{a}^l \in \mathbb{R}^{2d_{(l+1)}}$, and applying the LeakyRelu nonlinearity (with negative input slope $\alpha = 0.2$). Then attention coefficients are normalized across all choices of j using the softmax function:

$$\boldsymbol{\alpha}_{ij}^l = softmax_j(\mathbf{e}_{ij}) = \frac{exp(\mathbf{e}_{ij}^l)}{\sum_{k \in \mathcal{N}_i} exp(\mathbf{e}_{ik}^l)} \tag{7}$$

where $\mathcal{N}_i$ is the set containing node i and neighbors of node i. Once obtained, the normalized attention coefficients are used to compute a linear combination of the features corresponding to them, to serve as the final output features for each node:

$$\begin{aligned} \mathbf{Z}^0 &= \mathbf{X}, \\ \mathbf{Z}_i^{l+1} &= \sigma(\sum_{j \in \mathcal{N}_i} \boldsymbol{\alpha}_{ij}^l \mathbf{W}^l \mathbf{Z}_j^l) \end{aligned} \tag{8}$$

where $\sigma(\cdot)$ is a non-linear activation function, which is acted by exponential linear unit (ELU) [37] in our experiments. In particular, GAT applies the *multi-head attention mechanism*, which learns the embedding via Eq.(8) multiple times and concatenates the embedding into a new representation.

Graph Decoder Model: After we get the latent variable Z, we use a decoder model to reconstruct the graph structure A to better learn the relationship between two nodes. Here, the graph decoder model is defined by a generative model [35]:

$$p(A|Z) = \prod_{i=1}^{N} \prod_{j=1}^{N} p(A_{i,j}|\boldsymbol{z}_i, \boldsymbol{z}_j), \tag{9}$$

$$p(A_{ij} = 1|\boldsymbol{z}_i, \boldsymbol{z}_j) = \sigma(\boldsymbol{z}_i^T \boldsymbol{z}_j), \tag{10}$$

where A_{ij} are the elements of A and $\sigma(\cdot)$ denotes the logistic sigmoid function.
Optimization: To better learn class discriminative node representations, we optimize the variational graph autoencoder module via two losses as follows:

$$\mathcal{L}_S = \mathbb{E}_{q(Z|X,A)}[\log p(A|Z)] - KL[q(Z|X,A)||p(Z)] \tag{11}$$

where the first term is the reconstruction loss between the input adjacent matrix and the reconstructed adjacent matrix. The second term $KL[q(Z|X,A)||p(Z)]$ is the Kullback-Leibler divergence between $q(Z|X,A)$ and $p(Z)$, here $p(Z) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

4.2 Open-World Classifier Learning

After the variational graph autoencoder network, we obtain the uncertainty embeddings for each node through Eq. (4), which consists of two parts: uncertainty embeddings for labeled/training nodes $Z_{labeled}$ and uncertainty embeddings for unlabeled/test nodes $Z_{unlabeled}$. To better learn an accurate classifier for classifying both seen and unseen nodes in testing data, our proposed model consists of a cooperative module, a label loss as well as a class uncertainty loss working together to differentiate whether a node belongs to an existing class or an unseen class. The overall objective function is as follows:

$$\mathcal{L}_{OpenWGL} = \gamma_1 \mathcal{L}_L + \gamma_2 \mathcal{L}_C + \mathcal{L}_S \tag{12}$$

The γ_1, γ_2 are the balance parameters. The $\mathcal{L}_S$ is the loss function of the variational graph autoencoder network mentioned above. The $\mathcal{L}_L$ and $\mathcal{L}_C$ represent the label loss and the class uncertainty loss, respectively. The details are introduced as follows.
Label Loss: The label loss $\mathcal{L}_L$ is to minimize the cross-entropy loss for the labeled data:

$$\mathcal{L}_L(f_s(Z_{labeled}), Y) = -\frac{1}{N_l} \sum_{i=1}^{N_l} \sum_{c=1}^{C} y_{i,c} \mathsf{log}(\hat{y}_{i,c}) \tag{13}$$

In the above equation, $f_s(\cdot)$ denotes a full-connected layer with softmax activation function, where the full-connected layer is a linear transformation, transforming $Z_{unlabeled}$ into probabilities that sum to one. N_l is the number of labeled nodes. C denotes the number of seen classes, and $y_{i,c}$ denotes the

groundtruth of the i-th node in the labeled data, $\hat{y}_{i,c}$ is the classification prediction score for the i-th labeled node v_i in the c class, respectively.

Class Uncertainty Loss: Since we do not have the class information in the test data and there exists a considerable number of unseen nodes, we need to find a way to differentiate the seen class and unseen class. Unlike the label loss L_L, which can utilize the abundant training data and have a good performance on the seen class by the cross-entropy loss, the class uncertainty loss is proposed to balance the classification output for each node and have superior effects on the unseen nodes. In our paper, an entropy loss is placed as the class uncertainty loss and our goal is to **maximize** this entropy loss to make the normalized output of each node balanced. The formula is as follows:

$$\mathcal{L}_C(f_s(Z_{unlabeled})) = \frac{1}{N_u}\sum_{i=1}^{N_u}\sum_{c=1}^{C}\hat{y}_{i,c}\mathsf{log}(\hat{y}_{i,c}) \tag{14}$$

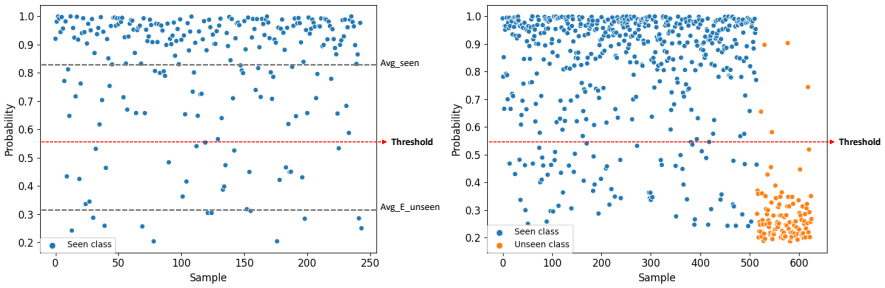
where N_u is the number of unlabeled nodes. $\hat{y}_{i,c}$ is the classification prediction score for the i-th unlabeled node v_i in the c class. Note that we do not put a negative sign in front of the formula as usual because we need to maximize the entropy loss. In addition, we will not use all the unlabeled data to maximize the entropy loss. We first sort all the unlabeled data output probability values (choosing the maximum probability for each node) after the softmax layer, and then discard the largest 10% (nodes with large probability values are easily classified into seen classes since their output is discriminative) and the smallest 10% nodes (nodes with small probability values means that the node's output is balanced over each seen class which can be easily detected as the unseen class). Finally the remaining nodes are utilized to maximize their entropy.

The training for label loss and class uncertainty loss acts like an adversarial process. On one hand, we want the label loss to influence the classifier to make the output of each node to be more discriminative and classify each seen node into correct classes via minimizing Eq. (13). On the other hand, we would like that the class uncertainty loss can make the output of each node to be more balanced to assist the detection of unseen class through the maximization of the entropy loss.

$\mathcal{L}_L$, $\mathcal{L}_C$ and $\mathcal{L}_S$ are jointly optimized via objective function defined in Eq. (12), and all parameters are optimized using the standard backpropagation algorithms.

4.3 Open-World Classification & Rejection

After performing the node uncertainty representation learning, we obtain a distribution (*i.e.*, the Gaussian distribution) of the node embeddings. Therefore, M different versions of feature vectors $(\boldsymbol{z}_i^1, \cdots, \boldsymbol{z}_i^M)$ are generated for each node v_i from this distribution via Eq. (4), where this process is called a reparametrization trick. Then the M different representations are fed into the softmax layer to turn them into probabilities over C classes respectively (each $\boldsymbol{z}_i^m$ can obtain an output vector $\boldsymbol{s}_i^m \in \mathbb{R}^{1\times C}$).



(a) In the validation set. (b) In the testing set.

Fig. 5: A visualization of determining the threshold using a validation set (only contains seen class instances). (a) determining the threshold using validation set. (b) applying determined threshold to the test set (contain both seen class and unseen class instances).

After this process, the M outputs are concatenated to obtain a sampling matrix $S_i \in \mathbb{R}^{M \times C}$. In S_i, each column denotes M different probabilities of a specific class and we average these probabilities for each class to obtain a vector $\boldsymbol{s}_{i,a} \in \mathbb{R}^{1 \times C}$. For the vector $\boldsymbol{s}_{i,a}$ with C different probabilities, we choose the largest one $\max(\boldsymbol{s}_{i,a})$. To recognize whether each node v_i is the seen or unseen classes for testing data, we have:

$$\hat{y} = \begin{cases} \text{Rejection}, & \text{if } \max_{c \in C} \ p(c|x_i) \leq t \\ \arg \ \max_{c \in C} \ p(c|x_i), & \text{otherwise.} \end{cases} \tag{15}$$

where $p(c|x_i)$ is obtained from the softmax layer output of $f_s(\cdot)$. If none of existing seen classes probability $p(c|x_i)$ value is above the threshold t, we reject x_i as a sample from the unseen class; otherwise, its predicted class is the one with the highest probability. The prediction process of each testing sample is illustrated in Fig. 4.

4.3.1 Automatic Rejection Threshold Selection

In open-world graph learning, a key problem is how to determine of the threshold t in Eq. (15) which can be used to reject a node from seen classes. In our paper, we propose a selection approach to automatically determine a threshold to reject nodes not belonging to seen classes. Specifically, we use a validation set X_{train}^{val}, which is separated from the training set X_{train} for threshold selection. For nodes in the validation set, we perform the node uncertainty representation learning, and conduct the same sampling process and choose the largest posterior probability. Then we average these chosen largest probabilities of all the nodes and obtain *avg_seen*. Because unseen class instances are assumed not appearing in the training set (including the validation set), we choose 10% nodes with the largest class distribution entropy, defined in

Eq. (16), as the "expected unseen class nodes" (ΔX^{val}_{train}), and their average posterior probability is denoted by avg_E_unseen.

$$H(x_i) = -\sum_{c \in C} p(c|x_i) \log p(c|x_i) \tag{16}$$

The final threshold is calculated by averaging the probabilities as follows,

$$t = \frac{avg\_seen + avg\_E\_unseen}{2} \tag{17}$$

Fig. 5(a) shows an example of the determining process in the validation set. We use this determined threshold to classify seen and unseen nodes in the test set in Fig. 5(b), and the result shows that the threshold is a good distinction between seen and unseen classes.

As a result of the above design, the node embedding features are denoted by distributions, instead of deterministic feature vectors. By using a sampling process to generate multiple versions of feature vectors, we are able to test the confidence of a node belonging to seen classes, and automatically determine a threshold to reject nodes not belonging to seen classes as unseen class nodes.

4.4 Algorithm Description

Our algorithm is illustrated in Algorithm 1. Given a graph $G = (V, E, X, Y)$, our goal is to obtain the node representations and classify the seen nodes and detect the unseen nodes, respectively. Firstly, we employ a variational graph autoencoder network to model the uncertainty of each node (Step 2-10). Here, the output Z is a distribution and we optimize the network through the KL loss and the reconstruction loss (Step 12). Then we propose two loss constraints $\mathcal{L}_L$ and $\mathcal{L}_C$ to make our model capable of classifying seen and unseen classes (Step 13-14). Finally, by jointly considering the label loss, class uncertainty loss and the VGAE loss (the KL divergence loss and network reconstruction loss), our model can better differentiate whether a node belongs to a seen class or an unseen class and capture the uncertainty representations for open-world graph learning.

4.5 Time Complexity Analysis

Given a graph $G = (V, E, X, Y)$ with N nodes (vertices), the proposed Open-World Graph Learning (OpenWGL) consists of two parts: graph encoder model and graph decoder model.

For GCN and GAT, the time complexity is asymptotically bounded by the number of edges of the network [29], *i.e.* $\mathcal{O}(|E|)$. This is mainly because that both methods rely on message passing between each node and its neighbors to learn node representation. E_i denotes edges incident to node v_i, for all nodes in the network, the total number of message passing is $\sum_{i=1}^{N} |E_i| =$

Algorithm 1: OpenWGL: Open-World Graph Learning

Date: $G = (V, E, X, Y)$: a Graph with links and features; $X = X_{train} \bigcup X_{test}$, $X_{test} = S \bigcup U$: S are the seen classes appeared in X_{train} and U are the unseen classes; C: the number of seen classes.
Result: $f(X_{test}) \mapsto Y$, $Y \in \{1, \cdots, C, rejection\}$.

while not convergence **do**
  // Graph Encoder Model
  For the first layer:
    $Z^{(1)} \leftarrow GNN(X, A)$
  For the second layer:
    $\boldsymbol{\mu} \leftarrow \mathsf{GNN}_{\mu}(Z^{(1)}, A)$
    $\log\boldsymbol{\sigma} \leftarrow \mathsf{GNN}_{\sigma}(Z^{(1)}, A)$
  $Z \leftarrow \boldsymbol{\mu} + \boldsymbol{\sigma} \cdot \zeta, \zeta \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  // Graph Decoder Model
  $p(A_{ij} = 1 | \boldsymbol{z}_i, \boldsymbol{z}_j) \leftarrow \sigma(\boldsymbol{z}_i^T \boldsymbol{z}_j)$
  // Compute Loss
  $\mathcal{L}_S \leftarrow$ Obtain the variational graph autoencoder loss using Eq. (11)
  $\mathcal{L}_L \leftarrow$ Obtain the label loss using Eq.(13)
  $\mathcal{L}_C \leftarrow$ Obtain the class uncertainty loss using Eq.(14)
  Back-propagate loss gradient using Eq.(12)
  $[W^{(1)}, W_{\mu}^{(2)}, W_{\sigma}^{(2)}, f_s(\cdot)] \leftarrow$ Update weights
  if early stopping condition satisfied **then**
    Terminate
For each test node $x \in X_{test}$:
  $f(x) \leftarrow$ classify x using Eq.(15).
return $f(X_{test})$

$2 \times |E| = \mathcal{O}(|E|)$. Because OpenWGL replies on graph encoder module, the time complexity of the graph encoder model is $\mathcal{O}(|E|)$. The time complexity of the process of reconstructing the original graph is $\mathcal{O}(dN^2)$, where d is the dimension of the latent space of matrix Z.

In order to test whether a node belongs to unseen class or not, OpenWGL needs to sample uncertain node embedding M times. Suppose the graph decoder model is sampled M times, the time complexity of the graph decoder model is $\mathcal{O}(dMN^2)$. As a result, the time complexity of OpenWGL is asymptotically bounded by $\mathcal{O}(|E| + dMN^2)$.

For sparse networks, the number of edges m is far less than the number of node pairs N^2, then the complex of OpenWGL is quadratic to the number of nodes $\mathcal{O}(|E| + dMN^2) = \mathcal{O}(dMN^2) = \mathcal{O}(N^2)$. For generic networks, the number of edges is less than the square of the number of nodes (the complete graph), *i.e.,*$|E| \leq N^2$, so we have $\mathcal{O}(|E| + dMN^2) = \mathcal{O}(N^2 + dMN^2) = \mathcal{O}((dM+1)N^2) = \mathcal{O}(N^2)$. In summary, the complexity of OpenWGL is $\mathcal{O}(N^2)$, which is mainly attributed to the graph encoder and decoder steps.

Table 1: Statistics of the benchmark datasets.

Dataset	# of Nodes	# of Edges	# of Features	# of Labels
Cora	2,708	5,429	1,433	7
Citeseer	3,312	4,732	3,703	6
DBLP	60,744	52,890	1,587	4
Pubmed	19,717	44,338	500	3

Table 2: Statistics of the number of nodes and number of classes of the benchmark data.

Dataset	Class						
	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
Cora	351	217	418	818	426	298	180
Citeseer	596	668	701	249	508	590	
DBLP	13586	23770	18292	5096			
Pubmed	4103	7739	7875				

5 Experiments

5.1 Experimental Setup

Benchmark Datasets We employ four widely used citation network datasets (Cora, Citeseer, DBLP, Pubmed) for node classification [38] [39]. The details of the experimental datasets are reported in Table 1.

Test Settings and Evaluation Metrics For each dataset, we hold out some classes as the unseen class for testing and the remaining classes as the seen classes. In Table 2, we report the statistics of the benchmark data, with respect to the number of nodes and number of classes. In the experiments, when the number of unseen classes are set as $|U| = 1$, $|U| = 2$, and $|U| = 3$, for Cora, Citeseer, and DBLP dataset, we select the last class, the last two classes, and the last three classes as the unseen class and the remaining classes as the seen classes of each dataset, respectively. For Pubmed dataset (with one unseen class $|U| = 1$), we select the first class as the unseen class and the remaining classes as the seen classes. We randomly sample 70% of nodes for training, 10% for validation and 20% for testing. Note that, the nodes of unseen class only appear in the testing set. We use the validation set to determine the threshold for rejecting the unseen class. Like the traditional semi-supervised node classification, for each dataset, we feed the whole graph into our model. We vary the number of unseen classes to verify the performance of our model at different unseen class proportion. We use the Macro F1 score and Accuracy for evaluation [1].

Baselines We employ following methods as baselines.

- GCN [32]: GCN is a deep convolutional network for graph-structured data. GCN employs a convolution layer to exploit the graph structure information and uses a classification loss function to guide the classification task. In GCN, it directly uses softmax as the final output layer. GCN does not have the rejection capability to the unseen class.
- GCN_Sigmod: In GCN_Sigmod, we use multiple 1-vs-rest of sigmoids rather than softmax as the final output layer of the GCN model, which also does not have the rejection capability to the unseen class.
- GCN_Sigmod_Thre: Based on GCN_Sigmod, we use the default probability threshold of $t_i = 0.5$ for classification of each class i, which means if all predicted probabilities are less than the threshold 0.5, we will reject it as the unseen class. Otherwise, its predicted class is the one with the highest probability.
- MLP_DOC: DOC [1] is the state-of-the-art open-world classification method for text classification. We use a two-layer perceptron to obtain the node representation.
- GCN_DOC: We utilize the rich node relationships and combine the GCN with DOC to compare with our model. In DOC, it uses multiple 1-vs-rest of sigmoids rather than softmax as the final output layer and defines an automatic threshold setting mechanism.

Proposed method: In order to validate the performance of the proposed OpenWGL learning algorithm, we implement OpenWGL using two types of graph neural networks, including Graph Convolutional Network (GCN) and Graph Attention Network (GAT).

- **OpenWGL_GCN:** OpenWGL_GCN employs a two-layer GCN as the graph encoder model to aggregate node features.
- **OpenWGL_GAT:** OpenWGL_GAT employs a two-layer GAT as the graph encoder model to aggregate node features.

All deep learning algorithms are implemented using Tensorflow [40,41] and are trained with Adam optimizer. We follow the evaluation protocol in open-world learning [1] [2] and evaluate all approaches through grid search on the hyperparameter space and report the best results of each approach. We feed the whole graph into our model when training. For all baseline methods, we use the same set of parameter configurations unless otherwise specified. For each deep approach, we use a fixed learning rate $1e^{-3}$. For each method, the GCNs contain two hidden layers ($L = 2$) with structure as $32 - 16$. The balance parameters γ_1, γ_2 are set to 1, 0.8, respectively. The dropout rate for each GCN layer is set to 0.3. M is set to 100. In addition, we choose 2 layers for OpenWGL_GAT, where the first GAT layer contains 32 hidden units, and the second layer contains 16 hidden units.

Table 3: Experimental results on Cora with different number of unseen classes $|U|$.

Methods	$\|U\| = 1$		$\|U\| = 3$	
	Accuracy	Macro F1	Accuracy	Macro F1
GCN	0.726	0.683	0.345	0.463
GCN_Sigmod	0.728	0.681	0.338	0.463
GCN_Sigmod_Thre	0.782	0.786	0.593	0.664
MLP_DOC	0.455	0.452	0.670	0.493
GCN_DOC	0.753	0.769	0.729	0.735
OpenWGL_GCN	**0.833**	**0.835**	**0.775**	**0.752**
OpenWGL_GAT	**0.843**	**0.845**	**0.818**	**0.786**

Table 4: Experimental results on Citeseer with different number of unseen classes $|U|$.

Methods	$\|U\| = 1$		$\|U\| = 3$	
	Accuracy	Macro F1	Accuracy	Macro F1
GCN	0.445	0.477	0.263	0.320
GCN_Sigmod	0.443	0.472	0.258	0.318
GCN_Sigmod_Thre	0.670	0.609	0.683	0.621
MLP_DOC	0.455	0.433	0.745	0.564
GCN_DOC	0.687	0.613	0.758	0.679
OpenWGL_GCN	**0.700**	**0.654**	**0.766**	**0.698**
OpenWGL_GAT	**0.702**	**0.658**	**0.767**	**0.700**

Table 5: Experimental results on DBLP with different number of unseen classes $|U|$.

Methods	$\|U\| = 1$		$\|U\| = 2$	
	Accuracy	Macro F1	Accuracy	Macro F1
GCN	0.662	0.562	0.285	0.323
GCN_Sigmod	0.662	0.562	0.290	0.323
GCN_Sigmod_Thre	0.657	0.650	0.282	0.326
MLP_DOC	0.643	0.630	0.480	0.477
GCN_DOC	0.657	0.658	0.503	0.506
OpenWGL_GCN	**0.688**	**0.689**	**0.653**	**0.642**
OpenWGL_GAT	**0.689**	**0.690**	**0.657**	**0.645**

5.2 Open-world Graph Learning Classification Results

Table 3, Table 4, Table 5 and Table 6 list the Macro F1 score and Accuracy of different methods on open-world node classification task. From the results, we have following observations:

(1) The GCN and GCN_Sigmoid obtain the worst performance among these baselines in all datasets since they do not have the rejection capability to the unseen class. Therefore, all the unseen nodes will be misclassified

Table 6: Experimental results on Pubmed with different number of unseen classes $|U|$. Note that, we only consider 1 class as unseen class since the Pubmed has 3 classes.

Methods	$\|U\| = 1$	
	Accuracy	Macro F1
GCN	0.480	0.429
GCN_Sigmod	0.483	0.427
GCN_Sigmod_Thre	0.513	0.498
MLP_DOC	0.586	0.595
GCN_DOC	0.631	0.640
OpenWGL_GCN	**0.753**	**0.757**
OpenWGL_GAT	**0.780**	**0.781**

Table 7: The Macro F1 score and Accuracy on three datatsets for closed-world settings (without unseen classes).

Dataset ($\|U\| = 0$)		GCN	OpenWGL_GCN	OpenWGL_GAT
Cora	Accuracy	0.863	0.854	0.854
	Macro F1	0.848	0.829	0.837
Citeseer	Accuracy	0.774	0.779	0.765
	Macro F1	0.752	0.745	0.742
DBLP	Accuracy	0.806	0.809	0.812
	Macro F1	0.754	0.751	0.753
Pubmed	Accuracy	0.867	0.863	0.869
	Macro F1	0.861	0.856	0.861

and their performance become worse with the number of unseen nodes increases.

(2) GCN_Sigmoid_Thre and GCN_DOC have better performances than GCN and GCN_Sigmoid, which shows that the threshold can improve the performance of detecting the unseen nodes. In addition, with the number of unseen nodes increases, GCN_Sigmoid_Thre and GCN_DOC become more competitive.

(3) GCN_DOC has better performance than GCN_Sigmoid_Thre in most cases, confirming that the threshold is not a fixed value and it varies with different datasets and the ratio of unseen class. DOC's automatic threshold setting mechanism can effectively improve the classification results of unseen class.

(4) The proposed Open-World Graph Learning model (OpenWGL_GCN and OpenWGL_GAT) consistently outperforms all baselines on three datasets with different numbers of unseen classes. It demonstrates that the proposed constrained graph variational encoder network can better differentiate whether a node belongs to a seen class or an unseen class and capture the uncertainty representation of each node by jointly considering the label loss, class uncertainty loss and the node uncertainty representation learning as a unified learning framework. In addition, the proposed OpenWGL_GAT outperforms OpenWGL_GCN which shows that assigning different weights

to nodes of a same neighborhood can be more beneficial for node representation learning.

(5) We also report closed-world learning setting results (without unseen class) in Table 7. The results show that when networks do not have unseen class, OpenWGL (OpenWGL_GCN and OpenWGL_GAT) has comparable performance as GCN, confirming its effectiveness and generalization for node classification. Overall, as the number of unseen classes increase, the performance of all methods, including our proposed methods, will decrease on Cora, DBLP, and Pubmed but increase on the Citeseer dataset. This is mainly because that as more nodes are being assigned as unseen class nodes, the network will have less label information, resulting in deterioration in performance. On the other hand, as more classes being treated as unseen class (*e.g.* from |U|=1 to |U|=3), the whole network will have less number of classes, resulting in a slightly higher random prediction accuracy (*e.g.* random prediction accuracy on a binary classification task is 50%, which is higher than 33.3%, the random prediction accuracy on a three class classification task). This possibly leads to the performance increase on the Citeseer dataset. Meanwhile, as the number of unseen classes increase, using thresholds to detect unseen class nodes has a clear benefits. However, in the absence of unseen classes, the performance of our method may be lower than rival methods, as shown in Table 7.

5.3 Ablation Analysis of OpenWGL Components

Because OpenWGL contains two key constraints, in this subsection, we compare variants of OpenWGL with respect to the following aspects to demonstrate: (1) the effect of the class uncertainty loss, and (2) the impact of the VGAE module (KL loss and reconstruction loss). Note that, we adopt GCN-based module in OpenWGL.

The following OpenWGL variants are designed for comparison.

- OpenWGL$\neg_C$: A variant of OpenWGL with only the class uncertainty loss being removed.
- OpenWGL$\neg_V$: A variant of OpenWGL with the KL loss and reconstruction loss being removed.

Tables 8, 9 & 10 report the ablation study results.

5.3.1 The effect of the class uncertainty loss

In order to show the superiority of the class uncertainty loss, we design a variant model OpenWGL$\neg_C$. As mentioned before, the class uncertainty loss is a constraint on the unlabeled nodes. The ablation study results show that the performances of the node classification task on both datasets are improved when the class uncertainty loss is used, indicating its effectiveness of detecting unseen nodes.

Table 8: The Macro F1 score and Accuracy between OpenWGL variants on Cora.

Methods	$\|U\| = 1$		$\|U\| = 3$	
	Accuracy	Macro F1	Accuracy	Macro F1
OpenWGL$^{\neg_C}$	0.782	0.787	0.700	0.665
OpenWGL$^{\neg_V}$	0.824	0.829	**0.785**	0.705
OpenWGL	**0.833**	**0.835**	0.775	**0.752**

Table 9: The Macro F1 score and Accuracy between OpenWGL variants on Citeseer.

Methods	$\|U\| = 1$		$\|U\| = 3$	
	Accuracy	Macro F1	Accuracy	Macro F1
OpenWGL$^{\neg_C}$	0.676	0.645	0.759	0.692
OpenWGL$^{\neg_V}$	0.691	0.648	0.760	0.683
OpenWGL	**0.700**	**0.654**	**0.766**	**0.698**

Table 10: The Macro F1 score and Accuracy between OpenWGL variants on DBLP.

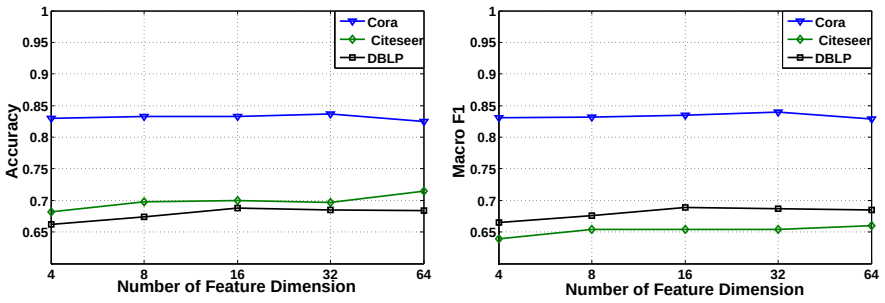
Methods	$\|U\| = 1$		$\|U\| = 2$	
	Accuracy	Macro F1	Accuracy	Macro F1
OpenWGL$^{\neg_C}$	0.675	0.672	0.650	0.631
OpenWGL$^{\neg_V}$	0.687	0.671	0.651	0.635
OpenWGL	**0.688**	**0.689**	**0.653**	**0.642**

5.3.2 The impact of the VGAE module (KL loss and reconstruction loss)

In order to verify the impact of the VGAE module which can model the uncertainty node representations, we compare OpenWGL model and OpenWGL$^{\neg_V}$. From the results, we can easily observe the OpenWGL model performs significantly better than OpenWGL$^{\neg_V}$. This confirms that the usage of KL loss can model the uncertainty to better capture the latent representation of each node, and reconstruction loss can preserve node relationships which will assist in the node representation.

5.4 Parameter Analysis

Impact of the feature dimensions of node output embeddings Z: As mentioned in the method section, the output of node embeddings is represented as Z. OpenWGL uses 2-layer GCNs with structure as $32 - 16$, and feature dimensions d of node output embeddings is 16. We vary d from 4 to 64 and report the results on three datasets, respectively in Fig. 6. On Citeseer and DBLP datasets, as d increases from 4 to 64, the performance grows gradually



(a) Accuracy (b) Macro F1

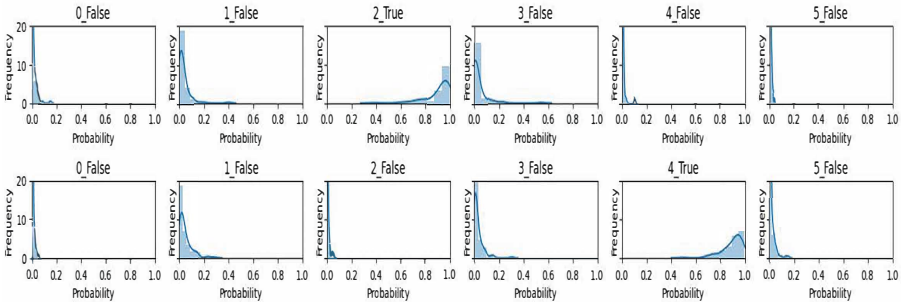
Fig. 6: Impact of feature dimensions of node output embeddings for the accuracy and Macro F1 score on three datasets.

to reach a plateau. The performance of Cora dataset is stable with d increasing from 4 to 32 and has a slight decrease at 64. When d further increases to 128, the accuracy of target domain is improved on both tasks. After that, both the Macro_F1 score and Accuracy remain steady and no obvious difference is observed with different d. Therefore, only slight differences can be observed with different d values. The increase of d, from 4 to 64, does not necessarily result in performance improvements. The results show that with sufficient feature dimensions ($d \geq 16$), OpenWGL is stable with the increasing number of feature dimensions.

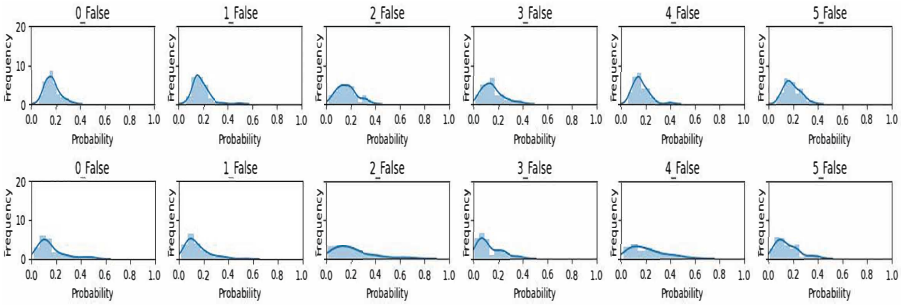
5.5 Case Study

5.5.1 Visualization of the OpenWGL sampling results

In order to verify the effectiveness of the sampling process of our model, we randomly choose two testing nodes from Cora dataset for seen and unseen classes (we choose one class as unseen, i.e. $|U| = 1$), respectively. After performing the node uncertainty representation learning, we obtain a distribution of the node embeddings. Then we generate 100 different versions of feature vectors for each node from this distribution and feed them into the softmax layer to turn them into probabilities over 6 classes, respectively. Therefore, after this process, for each node we obtain a 6×100 sampling matrix. In the sampling matrix, each column denotes 100 different probabilities of a specific class. We visualize the sampling matrices of these four nodes through histogram charts with seen and unseen classes in Figs. 7(a) and (b). In Fig. 7, each row represents one node and in each row, there are six subfigures indicating the 100 different probabilities of each class, respectively. From Fig. 7, we can observe that the sampling process have superior performance in differentiating the seen classes and the unseen class, and it is very helpful for determining the threshold. For example, as shown in the first row in Fig. 7(a), only in class 2, most of the 100 different probabilities are distributed on far right side of



(a) The visualization of two randomly selected nodes from seen classes.

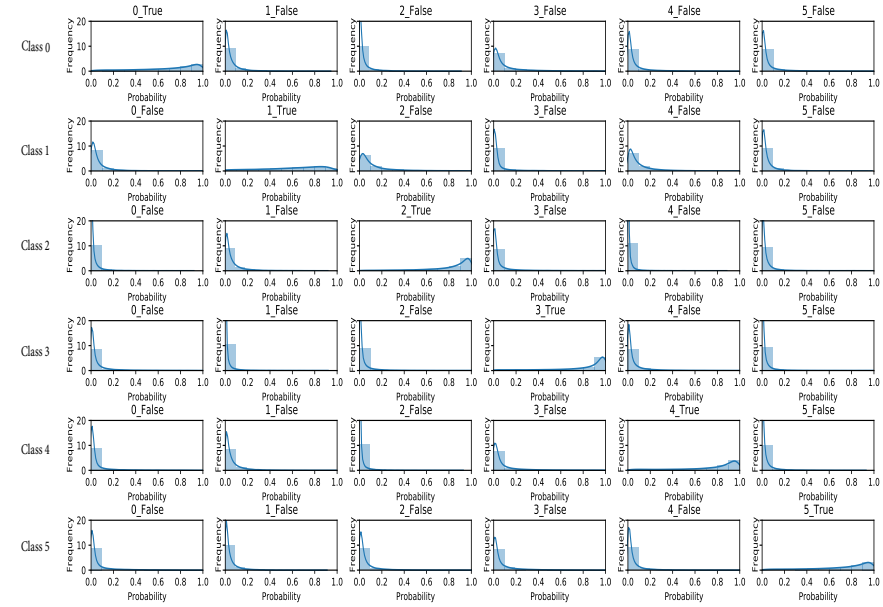


(b) The visualization of of two randomly selected nodes from unseen class.

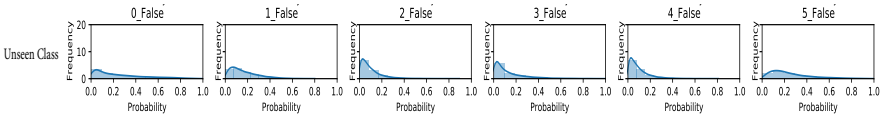
Fig. 7: A case study of the OpenWGL sampling results with two randomly selected nodes from seen classes and unseen class on Cora, respectively. Each row denotes one node, "True" denotes the class to which the node genuinely belongs, and "False" means that the node does not belong to this class. (a) two nodes randomly selected from seen classes, and (b) two nodes randomly selected from unseen class. The $x-$axis denotes the probability output of each node through the softmax classifier, and the $y-$axis denotes the frequency appearing in each class.

the histogram (i.e.,large probability), while all the other classes (0,1,3,4,5) are distributed on the far left side (i.e.,small probability). Thus, through the softmax layer, we can classify this node to class 2 and the ground truth is also class 2. However, if we just use a deterministic feature vector instead of this sampling method, this node may not be classified to the class 2, since class 2 also has cases with small probability values. Similarly, for the unseen nodes as shown in Fig. 7(b), in each seen class, most of the probability values are concentrated on the left side of the histogram (i.e., small probability), so we can easily detect them and classify them into unseen class. However, If we only obtain one probability output and do not have the sampling process, the unseen node might be misclassified randomly.

In order to show the average results of all nodes, Fig. 8 reports the sampling results using statistics of all nodes in each of the seen classes and unseen class on Cora, respectively. The difference between Fig. 7 and Fig. 8 is that the latter



(a) The visualization of statistics of all nodes in each of the classes (each row denotes a seen class).



(b) The visualization of statistics of all nodes of unseen class.

Fig. 8: A case study of the OpenWGL sampling results using statistics of all nodes of each seen classes and the unseen class on Cora, respectively. Each row denotes all nodes of each seen class, "True" denotes the class to which the the node belongs, and "False" means that the node does not belong to this class. (a) statistics of all nodes of each seen class, and (b) statistics of all nodes of unseen class. The $x-$axis denotes the probability output of each node through the softmax classifier, and the $y-$axis denotes the frequency of all nodes of per class appearing in different classes.

is obtained using the average of all nodes, whereas Fig. 7 is based on results from two randomly selected nodes. The results show that seen class nodes and unseen class nodes share different patterns. For seen class nodes, its average probability with respect to its genuine class is flat, with high probability values towards the 1.0 side, and its average probability with respect to other classes have high probabilities towards the 0.0 side. For unseen class nodes, it average probability values to all classes have high probabilities towards the 0.0 side, perfectly explain that the node does not belong to these classes.

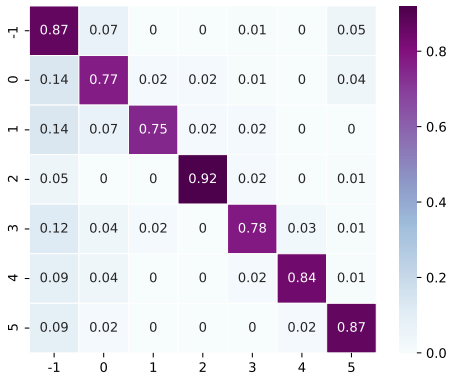


Fig. 9: The confusion matrix of OpenWGL on Cora. "-1" denotes the unseen class and "0,1,2,3,4,5" are seen classes. The (i, j) value of the matrix shows that the percentage value of the i-th class is classified to the j-th category.

5.5.2 The Confusion Matrix

In order to verify the effectiveness of OpenWGL in differentiating seen class nodes *vs.* unseen class nodes, Fig. 9 reports the confusion matrix of OpenWGL on Cora network, where "-1" denotes unseen class. The results show that OpenWGL correctly identifies 87% of unseen class nodes and also remains a high accuracy in classifying seen class nodes.

6 Discussion

Graph learning in an open-world setting is a significant challenge, because it involves feature learning, prediction loss, and classification confidence. In the proposed design, we combine multiple loss terms as objective function to learn embedding features to represent node for classification. This novel learning task has many interesting topics for future study.

In order to decide whether a node belongs to the unseen class, we use a thresholding approach, in Eq. (15), to reject a node from seen classes, if its posterior probability $p(c|x_i)$ is less than a threshold t. Although the threshold value t is automatically determined by Eq. (17), it is solely based on the posterior probability values $p(c|x_i), c \in C$. Alternatively, because rejecting x_i as seen class or not is a binary decision, one can design a binary classification task, by using features to learn whether instance x_i belongs to seen classes or not [2].

In this paper, we address the open-world graph learning in a static network setting, where nodes and edges do not change. In many applications, networks are continuously evolving with new nodes/edges [42], and node content many also change. Carrying out open-world graph learning in a dynamic network setting is another significant challenge. This is mainly because changes

in node/edge distributions may impact on the unseen class detection, and some seen class nodes may also be misclassified as unseen class if they are undergoing an evolving or concept drifting [43]. Finding good representation for nodes in dynamic networks, with capability to differentiate nodes in seen *vs.* unseen classes, is another topic for future study.

Currently, our method aims to attribute all unknown classes to an unseen classes (a single class), and it can not distinguish each unknown class. In the future, we will study to better distinguish different unknown classes through post-processing and other unsupervised methods, such as clustering.

In addition, we use two GNN variants (i.e., GAT and GCN) as Graph Encoder Model, and compares them empirically. Admittedly, there are other alternatives instead of GCN and GAT. However, in this paper, our goal is not to propose a novel graph representation learning model, but rather to focus on a new *open-world* graph learning paradigm, where the learning goal is to not only classify nodes belonging to seen classes into correct groups, but also classify nodes not belonging to existing classes to an unseen class. We also observe that OpenWGL_GAT outperforms OpenWGL_GCN which shows that new variant GNN method can be more beneficial for node representation learning, and can improve the performance of the model. We will try to apply some new GNN model, such as GIN [44], GIL [45], APPNP [46], and FiLMConv [47], for the *open-world* graph learning task in the future.

7 Conclusions

Traditional graph learning tasks are based on the closed-world setting, where unlabeled nodes (*i.e.* test set) should have the same class space as the labeled nodes (*i.e.* training set). The learning goal is to classify nodes into classes already known. In the paper, we advocated an open-world graph learning paradigm which not only classifies nodes belonging to seen classes into correct groups, but also classifies nodes not belonging to existing classes to an unseen class. To achieve the goal, we proposed an open-world graph learning (OpenWGL) framework with two major components: (1) node uncertainty representation learning, and (2) open-world classifier learning. The former uses label loss and class uncertainty loss to guide graph variational autoencoder to learn node embedding as distributions, and the latter automatically learns a threshold to detect unseen class nodes. The former learns a distribution for each node embedding via a graph variational autoencoder to capture the uncertainty, and the latter minimizes the label loss and class uncertainty loss simultaneously to distinguish seen and unseen class nodes, using automatically determined threshold. The threshold to reject the unseen class is further automatically determined in our framework. Experiments showed that when unseen class presents in test data, OpenWGL significantly outperforms baselines in classifying both seen and unseen class nodes. When networks do not have unseen class nodes (only contain nodes from seen classes), OpenWGL has a comparable performance to the baseline.

Acknowledgment

This research is supported by the U.S. National Science Foundation (NSF) through Grant Nos. IIS-1763452, CNS-1828181, and IIS-2027339.

References

1. L. Shu, H. Xu, and B. Liu, "Doc: Deep open classification of text documents," *arXiv preprint arXiv:1709.08716*, 2017.
2. H. Xu, B. Liu, L. Shu, and P. Yu, "Open-world learning and application to product classification," in *Proc. of WWW Conf.*, 2019, pp. 3413–3419.
3. G. Fei, S. Wang, and B. Liu, "Learning cumulatively to become more knowledgeable," in *Proc. of KDD*, 2016, pp. 1565–1574.
4. Z. Chen and B. Liu, "Lifelong machine learning," *Synthesis Lectures on Artificial Intel. and Machine Learning*, vol. 12, no. 3, pp. 1–207, 2018.
5. B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.
6. G. Fei and B. Liu, "Social media text classification under negative covariate shift," in *Proc. of EMNLP*, 2015, pp. 2347–2356.
7. W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boult, "Toward open set recognition," *IEEE Trans. on pattern analysis and machine intelligence*, vol. 35, no. 7, pp. 1757–1772, 2012.
8. W. J. Scheirer, L. P. Jain, and T. E. Boult, "Probability models for open set recognition," *IEEE Trans. on pattern analysis and machine intelligence*, vol. 36, no. 11, pp. 2317–2324, 2014.
9. L. P. Jain, W. J. Scheirer, and T. E. Boult, "Multi-class open set recognition using probability of inclusion," in *ECCV*. Springer, 2014, pp. 393–409.
10. M. Wu, S. Pan, and X. Zhu, "Openwgl: Open-world graph learning," in *Proc. of IEEE ICDM Conf.*, 2020.
11. Y. Gao, S. Chandra, Y. Li, L. Kan, and B. Thuraisingham, "Saccos: A semi-supervised framework for emerging class detection and concept drift adaption over data streams," *IEEE Trans. Knwl. & Data Eng.*, 2020.
12. X.-Q. C. Cai, P. Zhao, K.-M. Ting, X. Mu, and Y. Jiang, "Nearest neighbor ensembles: An effective method for difficult problems in streaming classification with emerging new classes," in *ICDM*, 2019.
13. X.-S. Wei, H.-J. Y. Ye, X. Wu, J. Wu, C. Shen, and Z.-H. Zhou, "Multiple instance learning with emerging novel class," *IEEE transactions knowledge and data engineering*, 2019.
14. G. Na, D. K. Kim, and H. Yu, "Dilof: Effective and memory efficient local outlier detection in data streams," in *Proc. of KDD*, 2019.
15. C. H. Park and H. Shim, "On detecting an emerging class," in *IEEE Intl. Conf. on Granular Computing (GRC 2007)*. IEEE, 2007, pp. 265–265.
16. B. Zadrozny and C. Elkan, "Transforming classifier scores into accurate multiclass probability estimates," in *Proc. of SIGKDD*, 2002.
17. M. Li and I. K. Sethi, "Confidence-based classifier design," *Pattern Recognition*, vol. 39, no. 7, pp. 1230–1240, 2006.
18. K. Proedrou, I. Nouretdinov, V. Vovk, and A. Gammerman, "Transductive confidence machines for pattern recognition," in *European Conference on Machine Learning*. Springer, 2002, pp. 381–390.
19. W. Soares-Filho, J. Seixas, and L. P. Caloba, "Enlarging neural class detection capacity in passive sonar systems," in *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No. 02CH37353)*, vol. 3. IEEE, 2002, pp. III–III.
20. E. M. Knorr and R. T. Ng, "Finding intensional knowledge of distance-based outliers," in *Vldb*, vol. 99, 1999, pp. 211–222.

21. L. Akoglu, H. Tong, and D. Koutra, "Graph based anomaly detection and description: a survey," *Data mining and knowledge discovery*, vol. 29, no. 3, pp. 626–688, 2015.
22. E. J. Spinosa and A. Carvalho, "Support vector machines for novel class detection in bioinformatics," *Genet Mol Res*, vol. 4:3, pp. 608–15, 2005.
23. D. Barbará, C. Domeniconi, and J. P. Rogers, "Detecting outliers using transduction and statistical testing," in *Proc. of KDD*, 2006, pp. 55–64.
24. M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *IJCNN*, vol. 2. IEEE, 2005, pp. 729–734.
25. F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.
26. M. Wu, S. Pan, C. Zhou, X. Chang, and X. Zhu, "Unsupervised domain adaptive graph convolutional networks," in *WWW '20: The Web Conference, April 20-24, 2020*, 2020, pp. 1457–1467.
27. M. Wu, S. Pan, X. Zhu, C. Zhou, and L. Pan, "Domain-adversarial graph neural networks for text classification," in *IEEE International Conference on Data Mining, ICDM*, 2019, pp. 648–657.
28. S. Zhu, L. Zhou, S. Pan, C. Zhou, G. Yan, and B. Wang, "GSSNN: Graph smoothing splines neural networks," in *AAAI*, 2020, pp. 7007–7014.
29. Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *TNNLS*, 2020.
30. S. Pan, R. Hu, S.-f. Fung, G. Long, J. Jiang, and C. Zhang, "Learning graph embedding with adversarial training methods," *IEEE Transactions on Cybernetics*, 2019.
31. M. Wu, S. Pan, L. Du, I. W. Tsang, X. Zhu, and B. Du, "Long-short distance aggregation networks for positive unlabeled graph learning," in *Proceedings of CIKM*, 2019, pp. 2157–2160.
32. T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
33. W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.
34. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.
35. T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.
36. D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.
37. D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.
38. C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information." in *Proc. of IJCAI*, 2015, pp. 2111–2117.
39. S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," in *Proc. of IJCAI*, 2016, pp. 1895–1901.
40. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/
41. J. Hu, S. Qian, Q. Fang, Y. Wang, Q. Zhao, H. Zhang, and C. Xu, "Efficient graph deep learning in tensorflow with tf_geometric," *CoRR*, vol. abs/2101.11552, 2021.
42. L. Chi, B. Li, X. Zhu, S. Pan, and L. Chen, "Hashing for adaptive real-time graph stream classification with concept drifts," *IEEE transactions on cybernetics*, vol. 48:5, pp. 1591–1604, 2018.
43. P. Zhang, B. J. Gao, X. Zhu, and L. Guo, "Enabling fast lazy learning for data streams," in *Proc. of IEEE ICDM Conference*, 2011, pp. 932–941.

44. K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.
45. S. Zhu, S. Pan, C. Zhou, J. Wu, Y. Cao, and B. Wang, "Graph geometry interaction learning," *Advances in Neural Information Processing Systems*, vol. 33, 2020.
46. J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.
47. M. Brockschmidt, "Gnn-film: Graph neural networks with feature-wise linear modulation," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119, 2020, pp. 1144–1152.