Machine Vision and Applications (2022) 33:4
https://doi.org/10.1007/s00138-021-01251-0

ORIGINAL PAPER

Graph neural networks in node classification: survey and evaluation

Shunxin Xiao[1] · Shiping Wang[1] · Yuanfei Dai[1] · Wenzhong Guo[1]

Received: 27 October 2019 / Revised: 8 September 2021 / Accepted: 23 September 2021 / Published online: 2 November 2021

Abstract
Neural networks have been proved efficient in improving many machine learning tasks such as convolutional neural networks and recurrent neural networks for computer vision and natural language processing, respectively. However, the inputs of these deep learning paradigms all belong to the type of Euclidean structure, e.g., images or texts. It is difficult to directly apply these neural networks to graph-based applications such as node classification since graph is a typical non-Euclidean structure in machine learning domain. Graph neural networks are designed to deal with the particular graph-based input and have received great developments because of more and more research attention. In this paper, we provide a comprehensive review about applying graph neural networks to the node classification task. First, the state-of-the-art methods are discussed and divided into three main categories: convolutional mechanism, attention mechanism and autoencoder mechanism. Afterward, extensive comparative experiments are conducted on several benchmark datasets, including citation networks and co-author networks, to compare the performance of different methods with diverse evaluation metrics. Finally, several suggestions are provided for future research based on the experimental results.

Keywords Deep learning · Graph neural networks · Node classification · Convolutional mechanism · Attention mechanism · Autoencoder mechanism

1 Introduction

With the rapid increase in computing resources and trainable data, the performance of neural networks has been excellent improved in various fields. In computer vision area, convolutional neural networks (CNNs) [1] are widely exploited for different research directions such as image classification [2,3], semantic segmentation [4,5] and image captioning [6,7]. In natural language processing area, recurrent neural networks (RNNs) [8] or long short-term network (LSTM) [9] are used for several important tasks including sentiment analysis [10,11], machine translation [12,13], question-answering system [14,15]. In addition, deep neural networks can also be used for cancer detection [16,17], earthquake magnitude prediction [18,19], or intelligent game [20].

Deep learning architectures have achieved great success in the Euclidean domain, including image, text and audio [21,22]. As one of the typical non-Euclidean structures in the machine learning area, graph data have the characteristics of arbitrary size, complex topological structure and always have an unfixed node ordering [23]. Therefore, it is difficult to directly exploit existing learning paradigms (e.g., convolutional or pooling operation) to graph structure data. However, graph data are a ubiquitous and essential structure in machine learning domain due to the powerful ability to represent objects and their relationships in various scenarios such as community detection [24,25], traffic flow prediction [26,27], knowledge graphs [28,29]. It is imperative to generalize these successful neural networks to graph analysis, and more and more researchers are devoting themselves to do this task [30]. As a result, graph neural networks (GNNs) have been developed rapidly and achieved a series of breakthroughs [31–33].

In the past few years, GNNs have been widely used in various graph analysis tasks, including node-focused tasks (e.g., node classification, link prediction) [34–36] and graph-focused tasks (e.g., graph similarity detection, graph classification) [37–39]. Among them, node classification is one of the most typical research directions of graph analysis due to the widespread application scenarios. In particular, the objective of the node classification task is to predict a particular class for each unlabeled node in the graph based on the

✉ Wenzhong Guo
guowenzhong@fzu.edu.cn

[1] College of Mathematics and Computer Sciences, Fuzhou University, Fuzhou, China

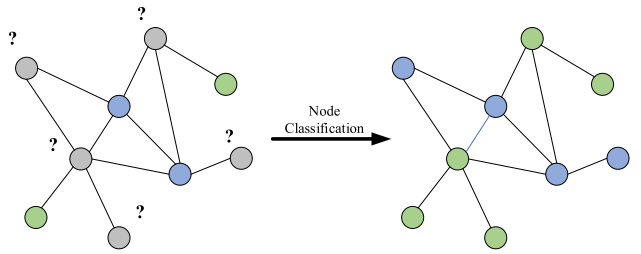

Fig. 1 Illustration of semi-supervised node classification. Blue and green denote those nodes that the label is known beforehand, and gray corresponds to the unlabeled nodes. The objective is to assign each gray node a label according to the information of those colorful nodes

graph information [40]. For example, node classification can predict the research topic to which each article belongs in the citation networks [41,42]. In the protein-protein interaction network, each node can be assigned several gene ontology types [43,44]. Figure 1 shows the objective of semi-supervised node classification in which only a few nodes have labels in the training dataset.

To provide a comparison of different algorithms in node classification, we present a review of graph neural networks based on comprehensive experiments. The contributions of this paper are shown as follows:

- This survey provides a comprehensive review of existing graph neural network models in the scene of node classification. It introduces a new taxonomy of these models and presents several popular algorithms for each category.
- Several popular algorithms derived from each category are compared based on a comprehensive experiment. In detail, these algorithms are rerunning on several popular benchmark datasets with four evaluation metrics. Besides, an objective analysis is conducted based on the experiment results.
- According to the experimental analysis, several challenges of existing GNNs models are discussed and several future research directions are provided for interested researchers.

The rest of this paper is organized as follows. In Sect. 2, we first define several commonly used notations and then introduce some definitions related to node classification. Then, various graph neural network models of different categories are introduced in Sect. 3. In Sect. 4, we conduct experiments about node classification by using graph neural networks on several datasets and discuss the experimental results. Based on the experimental analysis, several open problems and future directions are provided in Sect. 5. In Sect. 6, a conclusion of this survey is conducted.

Table 1 Summary of commonly used notations in this article. A list of notations including different variables and operations are provided in this table

Notations	Descriptions
$\mathcal{G}$	A graph
$\mathcal{V}$	The set of nodes
$\mathcal{E}$	The set of edges
v_i	A node and $v_i \in \mathcal{V}$
e_{ij}	An edge and $e_{ij} = (v_i, v_j) \in \mathcal{E}$
$\lvert \cdot \rvert$	The length of a list
n	The number of nodes and $n = \lvert\mathcal{V}\rvert$
m	The number of edges and $m = \lvert\mathcal{E}\rvert$
d	The dimension of input node feature
f	The dimension of output node vector
c	The number of node classes
$\mathbb{R}^m$	m-dimensional Euclidean space
$\mathbf{I}_n$	The identity matrix with dimension n
$\mathbf{X}$	The input node feature matrix
$\mathbf{X}_i$	The input feature of node v_i
$\mathbf{A}$	The adjacency matrix
$\mathbf{D}$	The diagonal degree matrix
$\mathbf{X}^T$	The transpose matrix of $\mathbf{X}$
$\mathbf{X}_{i,j}$	The i-th row, j-th column element
$\mathcal{L}$	A loss function
$\sigma(\cdot)$	A activation function
$\mathcal{N}(\cdot)$	A Gaussian distribution function
$N(i)$	The neighbors of node v_i
$[\cdot\|\|\cdot]$	Concatenation of matrices or vectors
$\odot$	Element-wise multiplication operation

2 Notations and definitions

In this section, we provide a set of commonly used notations and present some critical definitions that will be used in this article. Specifically, we provide a number of notations summarized in Table 1 before going further into the following contents.

The degree matrix $\mathbf{D}$ is derived from the corresponding adjacency matrix $\mathbf{A}$ and can be defined as

$$\mathbf{D}_{i,i} = \sum_{i \neq j} \mathbf{A}_{i,j}. \tag{1}$$

For other variables, we use lowercase italic characters to represent scalers, bold lowercase characters to denote vectors and bold uppercase characters to represent matrices. Then, several definitions are introduced in the following.

Definition 1 (graph) Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is a set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges between nodes in $\mathcal{V}$.

Note that $\mathcal{G}$ can be weighted or unweighted (the element's value of $\mathbf{A}$ is either 0 or 1), directed or undirected. In this paper, we focus on dealing with undirected graphs, which are augmented with the node attribute feature $\mathbf{X} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n\}$, where $\mathbf{X}_i$ is the corresponding feature vector of node v_i. In addition, for unattributed graphs, one-hot encoding is used as the features for all nodes, i.e., $\mathbf{X} = \mathbf{I}$.

Definition 2 (K-hop neighbors) Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, the k-hop neighbors of v_i denote the set of nodes which are exactly k hops away from v_i and is formulated as

$$N_k(i) = \{v_j | i \neq j, \min(\text{sp}(i, j), K) = k, \forall v_j \in \mathcal{V}\}. \quad (2)$$

Here, $\text{sp}(i, j)$ denotes the shortest path between v_i and v_j, and it will be infinite when there is no edge between them. K represents the maximum number of hops.

Definition 3 (node classification) Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, the objective of node classification is to produce labels for each node which can be formed as $\boldsymbol{y} = \{y_1, y_2, \ldots, y_n\}$ or $\mathbf{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n\}$ for multi-labels classification.

Note that the information of graph structure, node features, edge connections can be helpful to complete the node classification task.

3 Graph neural networks

In this section, we provide a comprehensive overview of various graph neural networks, which can be used for node classification, through three categories: convolutional mechanism, attention mechanism, and autoencoder mechanism. Several popular algorithms of each category are introduced.

3.1 Convolutional mechanism

Graph convolutional mechanism is one of the most commonly used information aggregation paradigms in graph analysis. The basic idea of this mechanism is that it utilizes convolutional or pooling operations on graph structure to extract higher representation for each node and then used in node classifier. GNNs models based on this mechanism are denoted as graph convolutional networks (GCNs), which are unrelated to node order and different from CNNs on images.

3.1.1 ChebNet

In order to generalize operators of CNNs to graph domain, Defferrard et al. [45] proposed a spectral-based graph convolutional network called ChebNet, which contains a fast localized spectral graph filter derived from Chebyshev polynomial. Specifically, ChebNet includes three main steps: the design of localized convolutional filters, a graph coarsening procedure, and a graph pooling operation.

Graph Laplacian is an important operator of spectral graph analysis [46] and can be defined as $\mathbf{L} = \mathbf{D} - \mathbf{A} \in \mathbb{R}^{n \times n}$. In addition, $\mathbf{L}$ has a normalized form which is formed as

$$\mathbf{L} = \mathbf{I}_n - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}. \quad (3)$$

The diagonalized form of graph Laplacian is defined as $\mathbf{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$, where $\boldsymbol{\Lambda}$ is the diagonal matrix of eigenvalues derived from $\mathbf{L}$ and $\mathbf{U}$ denotes the Fourier basis. $\text{diag}(\cdot)$ denotes a diagonal matrix derived from the input matrix or vector.

Then, a spectral filter g_θ is designed to filter the input signal $\mathbf{x} \in \mathbb{R}^n$ (each element associate with a node), which is shown as follows:

$$g_\theta(\mathbf{L})\mathbf{x} = g_\theta(\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T)\mathbf{x} = \mathbf{U}g_\theta(\boldsymbol{\Lambda})\mathbf{U}^T\mathbf{x}. \quad (4)$$

Here, $g_\theta(\boldsymbol{\Lambda}) = \text{diag}(\theta)$ is a nonparametric filter, $\theta \in \mathbb{R}^n$ denotes the Fourier coefficients, and $\mathbf{U}^T\mathbf{x} \in \mathbb{R}^n$ represents the graph Fourier transform of $\mathbf{x}$.

However, the nonparametric filter g_θ cannot localize in space and is complex to learn. To alleviate these problems, Defferrard et al. [45] introduce a polynomial filter and compute $g_\theta(\mathbf{L})$ in a recursive formulation based on the Chebyshev polynomial function. As a result, the filter can be parametrized as

$$g_\theta(\boldsymbol{\Lambda}) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\boldsymbol{\Lambda}}), \quad (5)$$

where $\tilde{\boldsymbol{\Lambda}} = 2\boldsymbol{\Lambda}/\lambda_{\max} - \mathbf{I}_n$ denotes a diagonal matrix which all elements in the range $[-1, 1]$, λ_{max} represents the maximum eigenvalue of $\mathbf{L}$ and the parameter $\theta \in \mathbb{R}^K$ denotes all coefficients of the Chebyshev polynomial $T_k(\mathbf{x})$. In detail, $T_k(\mathbf{x})$ can be computed in a recursive way $T_k(\mathbf{x}) = 2\mathbf{x}T_{k-1}(\mathbf{x}) - T_{k-2}(\mathbf{x})$, where $T_0(\mathbf{x}) = 1$ and $T_1(\mathbf{x}) = \mathbf{x}$. Thus, Eq. (4) can be reformed as follows:

$$g_\theta(\mathbf{L})\mathbf{x} = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{L}})\mathbf{x}, \quad (6)$$

where $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{\max} - \mathbf{I}_n$ is a scaled graph Laplacian and can then be used to evaluate the k-th order Chebyshev polynomial $T_k(\tilde{\mathbf{L}}) \in \mathbb{R}^{n \times n}$. Thus, the central vertex depends on its K-hop neighbors since Eq. (6) is a K-order polynomial in the Laplacian. Benefit from Eq. (6), ChebNet bypasses the computation of the graph Fourier basis.

Furthermore, Defferrard et al. [45] denoted $\overline{\mathbf{x}}_k = T_k(\tilde{\mathbf{L}})\mathbf{x} \in \mathbb{R}^n$ and the hidden state is recursively updated as $\overline{\mathbf{x}}_k =$

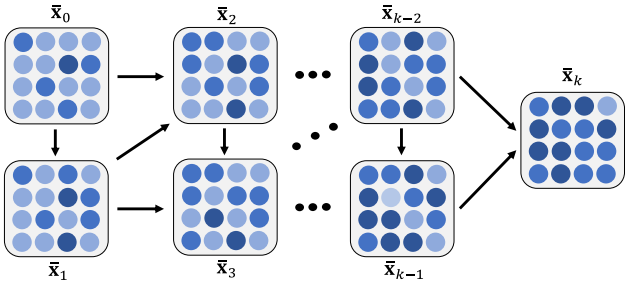

Fig. 2 Illustration of the iterative process used in ChebNet. The initial states $\bar{\mathbf{x}}_0$ and $\bar{\mathbf{x}}_1$ is set by $\bar{\mathbf{x}}_0 = \mathbf{x}$ and $\bar{\mathbf{x}}_1 = \tilde{\mathbf{L}}\mathbf{x}$, respectively. The final representation $\bar{\mathbf{x}}_K$ is obtained by several iterative computations

$2\tilde{\mathbf{L}}\bar{\mathbf{x}}_{k-1} - \bar{\mathbf{x}}_{k-2}$, where $\bar{\mathbf{x}}_0 = \mathbf{x}$ and $\bar{\mathbf{x}}_1 = \tilde{\mathbf{L}}\mathbf{x}$. This iterative process is shown in Fig. 2. Finally, the complete filter can be defined as $\mathbf{z} = g_\theta(\mathbf{L})\mathbf{x} = \left[\bar{\mathbf{x}}_0, \bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_{K-1}\right]\theta$ which contains $\mathcal{O}(Km)$ operations.

To provide a meaningful graph for pooling operation, similar nodes should be clustered together where this process can be regarded as graph clustering, which is NP-hard. Thus, Defferrard et al. [45] utilized the Graclus multi-level clustering algorithm [47] to aggregate similar nodes and generate a coarsening graph. However, the order of vertices is not arranged in a meaningful way after coarsening. Thus, it needs to store all matched vertices when applying pooling operation, which would cause inefficient memory and computation. In order to accelerate the pooling operations, Defferrard et al. [45] proposed a pooling strategy that has the same efficiency as a 1D pooling. In detail, the procedure contains two steps: (i) construct a balanced binary tree and (ii) rearrange the nodes. As a result, it makes the pooling fast and suitable for parallel architectures.

3.1.2 GCN

Instead of using the information derived from K-hop neighbors to represent each node introduced in ChebNet [45], Kipf and Welling [48] also propose a spectral-based graph convolutional network (GCN) that encapsulates the hidden representation of each target node by aggregating the feature information of its first-order approximate neighbors [49]. Then, a deep neural network model is built by stacking such graph convolutional layers multi-times and is then used to obtain the final hidden representation of each node. As a result, the learned representation also contains information of its multi-hop neighbors similar to ChebNet [45].

Specifically, Kipf and Welling [48] set the number of hops as $K = 1$. Therefore, Eq. (6) becomes a linear function and is reformed as

$$\begin{aligned}\mathbf{z} &= \theta_0 T_0(\tilde{\mathbf{L}})\mathbf{x} + \theta_1 T_1(\tilde{\mathbf{L}})\mathbf{x} \\ &= \theta_0\mathbf{x} + \theta_1\Big(\frac{2}{\lambda_{max}}\mathbf{L} - \mathbf{I}_n\Big)\mathbf{x},\end{aligned} \tag{7}$$

with only two parameters θ_0 and θ_1. Furthermore, Kipf and Welling [48] set the maximum eigenvalue as a constant $\lambda_{max} = 2$ and used a unique parameter $\theta = \theta_0 = -\theta_1$ to address the problem of overfitting on the local structure of a graph and minimize the number of operations. Under this setting and Eqs. (3), (7) is reformed as

$$\mathbf{z} = \theta\left(\mathbf{I}_n + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)\mathbf{x}. \tag{8}$$

Here, all eigenvalues of expression $\mathbf{I}_n + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ are lying in $[0, 2]$ and the parameters of the filter are shared by all layers.

Note that stacking such convolutional operator to build a deep neural network model might produce some problems such as numerical instabilities and exploding/vanishing gradients. To solve these problems, Kipf and Welling [48] used a *renormalization trick* strategy for the above expression as

$$\mathbf{I}_n + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \rightarrow \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}, \tag{9}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_n$ is the adjacency matrix of the graph with a self-connection. Similar to Eq. (1), $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$ is the degree matrix with respect to $\tilde{\mathbf{A}}$.

Then, Kipf and Welling [48] generalized the definition of Eq. (9), which is used for input signal $\mathbf{x} \in \mathbb{R}^n$ with only one channel, to the situation where each signal has multiple channels $\mathbf{X} \in \mathbb{R}^{n\times d}$. Here, d denotes the number of input channels (i.e., the dimension of node feature vector). To this end, the convolutional filter for signal $\mathbf{X}$ is defined as follows:

$$\mathbf{Z} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{W}. \tag{10}$$

Here, $\mathbf{W} \in \mathbb{R}^{d\times f}$ is the matrix of filter parameters, $\mathbf{Z} \in \mathbb{R}^{n\times f}$ is the convolved feature matrix for all nodes and f denotes the dimensional of the embedding feature.

After defining the convolutional filter of each layer, Kipf and Welling [48] intent to build a multi-layer GCN with a layer-wise propagation rule:

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right), \tag{11}$$

where $\mathbf{W}^{(l)}$ is a trainable weighted matrix, $\mathbf{H}^{(l)} \in \mathbb{R}^{n\times h}$ is the matrix of hidden states for the l-th layer, and h denotes the dimension of higher representation. $\mathbf{H}^{(0)}$ is initialized by using the input signal $\mathbf{X}$.

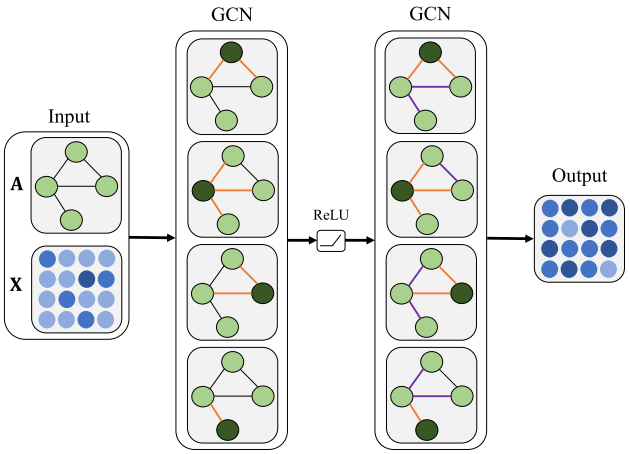

Fig. 3 Schematic diagram of a two-layer GCN model. The dark green denotes target nodes that need to aggregate information from neighbors. The orange and purple lines denote the information stream for first-hop and second-hop neighbors, respectively. As a result, the output contains both first-hop and second-hop information

For semi-supervised node classification, Kipf and Welling [48] proposed a two-layer GCN model, which is schematically depicted in Fig. 3. In practice, Kipf and Welling [48] computed $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ in pre-processing phase and introduced a forward propagation model as follows:

$$\begin{aligned}\mathbf{Z} &= f(\mathbf{X}, \mathbf{A}) \\ &= \sigma\left(\hat{\mathbf{A}}\,\mathrm{ReLU}\left(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)}\right)\mathbf{W}^{(1)}\right),\end{aligned} \tag{12}$$

where $\mathbf{W}^{(0)} \in \mathbb{R}^{d\times h}$ is a matrix that maps the input feature to the hidden representation, $\mathbf{W}^{(1)} \in \mathbb{R}^{h\times f}$ is a hidden-to-output matrix, and $\sigma(\cdot)$ is implemented by a softmax function.

3.1.3 GraphSAGE

However, ChebNet and GCN are inherently transductive that need all nodes are existing during the training process and cannot generalize to previously unseen nodes. In order to enable a model to become inductive that has the ability to deal with those unseen nodes, Hamilton et al. [50] proposed a spatial-based graph convolutional network called GraphSAGE (SAmple and aggreGatE), which utilizes both the feature information of nodes (e.g., the TF-IDF feature when one node represents for one document) and the structural attributes of a node's local neighborhood to learn a higher representation for each node.

Instead of training different hidden representations for each node separately, Hamilton et al. [50] designed a group of aggregators to learn embeddings by combining the information of nearby nodes of the current central node. Then, these

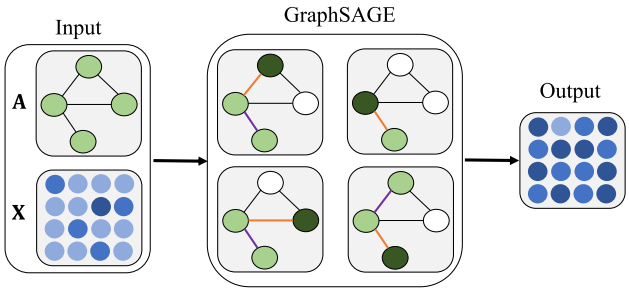

Fig. 4 Illustration of GraphSAGE model with a sampling strategy. Here, the maximum number of neighbors for each hop is set as one and the maximum hop is set as two. The orange and purple edges denote different aggregators. After sampling, the information of white nodes are not used in the aggregation stage

aggregators are combined to form the forward propagation algorithm of GraphSAGE. There are existing K aggregators, which can be denoted as $\mathrm{AGGREGATE}_k, \forall k \in \{1, \ldots, K\}$, and K parameter matrices $\mathbf{W}^k, \forall k \in \{1, \ldots, K\}$, which are used as the transformer between different hops.

In order to obtain the hidden state $\mathbf{h}_i^k$ of each target node v_i in each step $k \in \{1, \ldots, K\}$, Hamilton et al. [50] firstly used $\mathrm{AGGREGATE}_k$ to produce the aggregated neighborhood vector $\mathbf{h}_{N(i)}^k$ by using the information of all immediate neighbors of v_i which was produced in the previous time step. Then, $\mathbf{h}_{N(i)}^k$ is concatenated with the previous hidden state $\mathbf{h}_i^{k-1}$ of node v_i, and this concatenated vector is then transformed by $\mathbf{W}^k$ with an activation function to form the current state of the target node v_i. Repeats the above process, the final feature representation of each node $\mathbf{Z}_i$ is produced in the K-th step.

In order to reduce the complexity of GraphSAGE and scale to a large graph, Hamilton et al. [50] utilized a sample strategy to choose a fixed-size set of neighbors for each target node. The illustration of the GraphSAGE model with a sampling strategy is shown in Fig. 4.

Therefore, the complexity of GraphSAGE becomes $O\left(\prod_{k=1}^{K} S_k\right)$, where S_k denotes the size of the neighbor's set in the k-th step.

Additionally, Hamilton et al. [50] introduced three different aggregators, including mean aggregator, LSTM aggregator and pooling aggregator. In the mean aggregator, the aggregated neighbor vector $\mathbf{h}_{N(i)}^k$ is derived by simply utilizing the element-wise mean operation on the hidden state of neighbor $\left\{\mathbf{h}_j^{k-1}, \forall v_j \in N(i)\right\}$, which can be formed as

$$\mathbf{h}_{N(i)}^k \leftarrow \sigma\left(\mathbf{W} \cdot \mathrm{MEAN}\left(\mathbf{h}_j^{k-1}, \forall v_j \in N(i)\right)\right) \tag{13}$$

Equation (13) is similar to the convolutional operator in the GCN [48] and can be modified to an inductive variant from the original transductive framework. In the LSTM aggregator, Hamilton et al. [50] simply used an unordered sequence

of node's neighbors as the input of an LSTM model. In the pooling method, an element-wise max-pooling operator is used to gather information for each node, which is shown as follows:

$$\mathbf{h}_{N(i)}^{k} \leftarrow \max\left(\left\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_{j}^{k}+\mathbf{b}\right), \forall v_j \in N(i)\right\}\right), \quad (14)$$

where max represents the pooling operator. Note that the mean and pooling aggregator are symmetric means that they are permutation invariant.

3.1.4 APPNP

Recently, Xu et al. [51] proposed an algorithm to reduce the complexity of GNNs by using the relationship between a random walk algorithm and a common message passing algorithm. However, this model considers a graph as a whole and loses the ability to capture the different information of local neighbors for each starting root node. Instead of using the random walk, Klicpera et al. [52] utilized the relationship between GCN and PageRank [53] to formulate a personalized PageRank-based propagation architecture. This propagation scheme is then used to design a graph convolution-based network called personalized propagation of neural prediction (PPNP).

In detail, Klicpera et al. [52] defined the personalized PageRank for each starting node v_i by using a recursive formulation, which is shown as follows:

$$\boldsymbol{\pi}_{\text{ppr}}\left(\mathbf{R}_i\right) = (1-\alpha)\hat{\mathbf{A}}\pi_{\text{ppr}}\left(\mathbf{R}_i\right) + \alpha\mathbf{R}_i, \quad (15)$$

where $\mathbf{R}_i$ is a one-hot indicator vector that denotes the teleport vector of node v_i, and $\alpha \in (0, 1]$ is the teleport (or restart) probability that can be used to control the range of neighbor for each target node. Thus, Eq. (15) can be solved as the following form

$$\boldsymbol{\pi}_{\text{ppr}}\left(\mathbf{R}_i\right) = \alpha\left(\mathbf{I}_n - (1-\alpha)\hat{\mathbf{A}}\right)^{-1}\mathbf{R}_i. \quad (16)$$

Note that $\mathbf{R}_i$ is helpful to preserve the information of a node's local neighbors. Further, Klicpera et al. [52] defined the fully personalized PageRank matrix by stacking all $\boldsymbol{\pi}_{\text{ppr}}\left(\mathbf{R}_i\right)$ and replacing it with an identity matrix. The final representation of this PageRank matrix is shown as

$$\boldsymbol{\Pi}_{\text{ppr}} = \alpha\left(\mathbf{I}_n - (1-\alpha)\hat{\mathbf{A}}\right)^{-1}. \quad (17)$$

Here, each element of $\boldsymbol{\Pi}_{\text{ppr}}$ represents the influence score between two nodes. For example, the influence score of node v_i on node v_j is proportional to the value of $\boldsymbol{\Pi}_{\text{ppr}}^{(ji)}$ and can be defined as $I(i, j) \propto \boldsymbol{\Pi}_{\text{ppr}}^{(ji)}$.

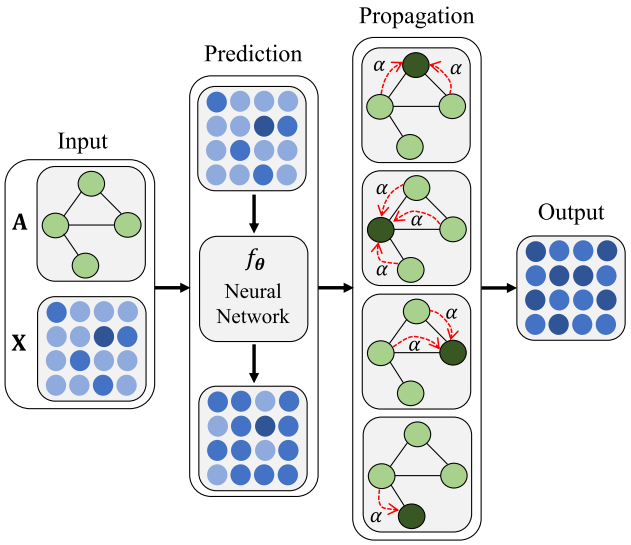

Fig. 5 Illustration of the PPNP model. First, a neural network is used to generate predictions. Then, an adaptation of personalized PageRank is utilized to propagate information between samples. α represents the teleport (or restart) probability. The self-connection is not considered

The process of the PPNP model can be divided into two consecutive parts, prediction and propagation, which is illustrated by Fig. 5. In the prediction stage, PPNP uses a neural network f_θ with parameter set θ to generate predictions for each node based on the input node features $\mathbf{X}$. In the propagation step, the predictions matrix $\mathbf{H} \in \mathbb{R}^{n\times f}$ is used to produce the final result via the fully personalized PageRank scheme. So, the equation of PPNP can be defined as

$$\mathbf{Z} = \sigma\left(\alpha\left(\mathbf{I}_n - (1-\alpha)\hat{\mathbf{A}}\right)^{-1}\mathbf{H}\right), \quad (18)$$

where $\mathbf{H}_i = f_\theta(\mathbf{x}_i)$ and $\sigma(\cdot)$ is implemented by a softmax function. Note that the propagation matrix $\hat{\mathbf{A}}$ can be replaced with other forms such as $\mathbf{A}_{\text{rw}} = \mathbf{A}\mathbf{D}^{-1}$. Furthermore, the number of propagation layers is arbitrary (in fact, infinitely many) and can avoid the oversmoothing problem.

The computational complexity and memory requirement of the PPNP model is $\mathcal{O}\left(n^2\right)$. It is inefficient and may cause the outing of memory more easily for a large graph. In order to solve this problem, Klicpera et al. [52] proposed an approximate personalized propagation of neural predictions (APPNP) by replacing the full personalized PageRank with a variant of topic-sensitive PageRank [54]. In APPNP, the final predictions of all nodes are calculated via some propagation layer by a recursive formulation, which is shown as

$$\mathbf{Z}^{k+1} = (1-\alpha)\hat{\mathbf{A}}\mathbf{Z}^{k} + \alpha\mathbf{H}, \quad (19)$$

where $\mathbf{Z}^0 = \mathbf{H} = f_\theta(\mathbf{X})$ and the output of the last layer is followed by a softmax function.

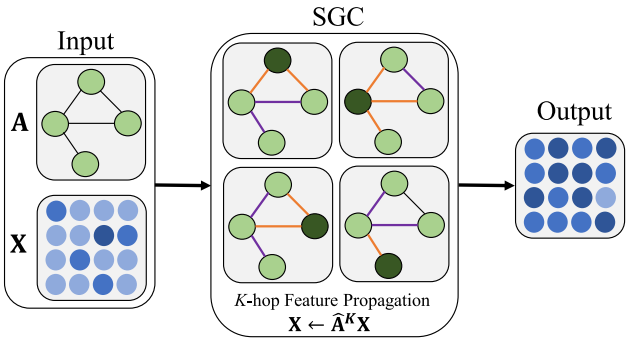

Fig. 6 Illustration of SGC. Here, the SGC model reduces the entire procedure of a multi-layer model to a single feature propagation step. However, the final output also contains both first-hop and second-hop information in this example

3.1.5 SGC

The development of traditional neural networks is from simplicity to complexity, e.g., from the logistic regression to CNNs for grid-like data or RNNs used in the sequence-like data. However, the computational complexity and memory requirement of most existing GNNs are always higher even in the early algorithms [48] and do not follow the trend that appears in the development of neural networks used for Euclidean data. In order to reduce the complexity of existing models, Wu et al. [55] proposed a linear graph neural network called simple graph convolutional (SGC) which is illustrated in Fig. 6.

The SGC model holds a hypothesis that the improved performance of GCNs is derived from the usage of information aggregated from neighborhoods.

Under this assumption, Wu et al. [55] removed all nonlinear transition functions except the softmax function in the original GCNs to reduce the cost of computing. Therefore, the definition of this linear network can be obtained by modifying Eq. (12) and is shown as follows:

$$\mathbf{Z} = \sigma(\hat{\mathbf{A}} \ldots \hat{\mathbf{A}}\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^0\mathbf{W}^1 \ldots \mathbf{W}^K), \tag{20}$$

where $\sigma(\cdot)$ is implemented by a softmax function. Note that Eq. (20) can aggregate information from K-hop neighbors similar to the K-layer GCN model. Further, Wu et al. [55] collapse all the weight matrices between different GCN layers into a single matrix $\mathbf{W} = \mathbf{W}^0\mathbf{W}^1 \ldots \mathbf{W}^K$. Similarly, the repeated multiplication of $\hat{\mathbf{A}}$ is collapsed into K power $\hat{\mathbf{A}}^K$. After this simplify, the definition of SGC can be reformed as

$$\mathbf{Z} = \sigma(\hat{\mathbf{A}}^K\mathbf{X}\mathbf{W}), \tag{21}$$

where $\hat{\mathbf{A}}^K$ is a low-pass-type filter and calculated in the preprocessing stage. SGC is naturally to interpret because of the dividing of feature extraction $\mathbf{H} = \hat{\mathbf{A}}^K\mathbf{X}$ and classification $\mathbf{Z} = \sigma(\mathbf{H}\mathbf{W})$. The latter is implemented by logistic regression and trained with any second-order method or stochastic gradient descent [56].

3.2 Attention mechanism

Attention mechanism is one of the most useful architecture in artificial intelligence, including computer vision, graph analysis. Instead of using a constant weight, different neighbors should have variant contributions for the target node. Additionally, the number of neighbors is variant to each node. The benefits of attention mechanism are dealing with variable-sized inputs and focusing on the most relevant part. Thus, it is natural to apply attention mechanism in node classification.

3.2.1 GAT

In order to capture different importance of neighborhood for a target node when producing the higher hidden representation, Veličković et al. [57] introduced an attention-based graph convolutional layer and is then used to construct arbitrary deep graph attention networks (GAT) by stacking such attention layers.

On the whole, the objective of the GAT model is to produce a new set of features $\mathbf{Z} = \{\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_n\} \in \mathbb{R}^{n\times f}$ for all nodes by using $\mathbf{X} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n\}$ as the model inputs. In detail, GAT firstly utilizes a learnable linear transformer to obtain the hidden representations $\mathbf{H}_i = \mathbf{W}\mathbf{X}_i \in \mathbb{R}^f$ for each node v_i. Here, $\mathbf{W} \in \mathbb{R}^{f\times d}$ denotes the parametrized weight matrix of this transformer. Then, Veličković et al. [57] designed an attentional function ATTENTION : $\mathbb{R}^f \times \mathbb{R}^f \rightarrow \mathbb{R}$, to compute attention weights between each pair of nodes by using the above hidden representation

$$e_{ij} = \text{ATTENTION}\left(\mathbf{H}_i, \mathbf{H}_j\right). \tag{22}$$

Specifical, e_{ij} denotes the importance of node v_j when computing the representation of node v_i. Note that, Veličković et al. [57] only use the first-order appropriate neighbors in Eq. (22). In order to make the weight coefficients more comparable across different nodes, a softmax function is used to normalize e_{ij} and the improved form is defined as

$$\alpha_{ij} = \text{softmax}_j\left(e_{ij}\right) = \frac{\exp\left(e_{ij}\right)}{\sum_{v_k\in N(i)}\exp\left(e_{ik}\right)}. \tag{23}$$

In fact, Veličković et al. [57] only used a single forward neural network with an activation function to compute the normalized coefficients. Thus, Eq. (23) is reformed as

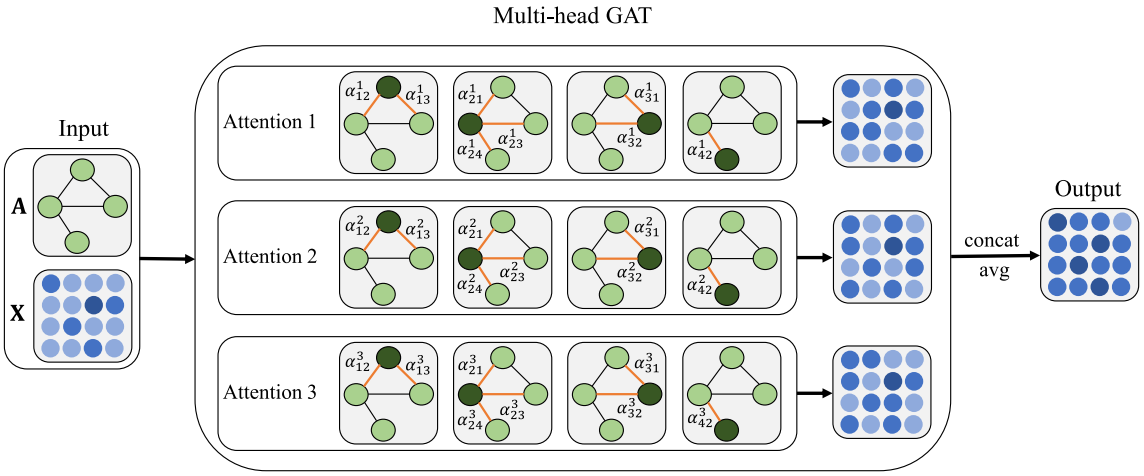

Fig. 7 Illustration of multi-head attention mechanism used in the GAT model. Here, the number of attention is set as $K = 3$. The aggregated features from each head are concatenated or averaged to obtain a higher representation for each node. The average operation is only used in the output layer. The self-connection is not considered

$$\alpha_{ij} = \frac{\exp\left(\sigma\left(\mathbf{a}^T\left[\mathbf{H}_i \parallel \mathbf{H}_j\right]\right)\right)}{\sum_{v_k \in N(i)} \exp\left(\sigma\left(\mathbf{a}^T\left[\mathbf{H}_i \parallel \mathbf{H}_k\right]\right)\right)}, \tag{24}$$

where $\mathbf{a} \in \mathbb{R}^{2f}$ is the weight vector of the proposed attention mechanism implemented by a single-layer feedforward network and $\sigma(\cdot)$ is implemented by function LeakyReLU. Once the coefficients of a node's neighbors are all computed, a linear combination for all neighbors is used to obtain the final representation for each target node which is shown as follows:

$$\mathbf{Z}_i = \sigma\left(\sum_{v_j \in N(i)} \alpha_{ij}\mathbf{H}_j\right). \tag{25}$$

Furthermore, Veličković et al. [57] applied a multi-head attention mechanism to improve the stability of the learning process, which is inspired by Vaswani et al. [58]. This improved mechanism is illustrated in Fig. 7 and can be defined as

$$\mathbf{Z}_i = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{v_j \in N(i)} \alpha_{ij}^k \mathbf{W}^k \mathbf{X}_j\right), \tag{26}$$

where α_{ij}^k denotes the normalized coefficients computed by the k-th attention function ATTENTION_k and $\parallel$ represents the operator which concatenating the output feature of all attention functions. Note that the concatenate operator of Eq. (26) will be replaced with an averaging operator if the multi-head attention is used in the last layer, and then, Eq. (26) can be redefined as

$$\mathbf{Z}_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in N(i)} \alpha_{ij}^k \mathbf{H}_j^k\right). \tag{27}$$

Here, $\mathbf{H}_j^k = \mathbf{W}^k\mathbf{X}_j$ is computed by the k-th linear transformer. Additional, GAT is efficient due to the parallelized compute of the attention layer and the complexity of one layer is appropriate to $\mathcal{O}(ndf + mf)$.

3.2.2 AGNN

Similarly, Kiran et al. [59] proposed a new attention-based graph neural network (AGNN) with a dynamic and adaptive propagation layer and then used it to deal with the situation where the supervised information in the original data is scarce.

The AGNN model consists of one word-embedding layer and several attention-guided propagation layers. The front is used to map a bag-of-words representation of a document into an averaged word embedding. In the word-embedding layer, a linear transformer followed by an activation function is used to derive the initially hidden representation for each node by using the input features. Thus, the first layer of the AGNN model can be defined as $\mathbf{H}^0 = \sigma\left(\mathbf{X}\mathbf{W}^0\right)$, where $\mathbf{W}^0 \in \mathbb{R}^{d\times h}$ denotes the transform matrix, and $\sigma(\cdot)$ is implemented by the rectified linear unit (ReLU) that can be calculated as $\text{ReLU}(x) = \max(0, x)$. In fact, $\mathbf{H}^0 \in \mathbb{R}^{n\times h}$ is a hidden matrix by stacking the hidden state of all nodes.

Then, the normalized attention coefficient from node v_j to node v_i in the k-th layer is computed as follow

$$\alpha_{ij}^{k} = \frac{\exp\left(\beta^{k}\cos\left(\mathbf{H}_{i}^{k}, \mathbf{H}_{j}^{k}\right)\right)}{\sum_{v_r \in N(i)\cup\{v_i\}} \exp\left(\beta^{k}\cos\left(\mathbf{H}_{i}^{k}, \mathbf{H}_{r}^{k}\right)\right)}. \tag{28}$$

Here, $\cos(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}/\|\mathbf{x}\|\|\mathbf{y}\|$ with an L_2 norm, e.g., $\|x\|$ and $\beta^k \in \mathbb{R}$ is the only scalar parameter of k-th attention-guided propagation layer. Note that in Eq. (28), the self-connected attention coefficient is also considered. Similar to the work produced by Veličković et al. [57], the coefficient of a node's neighbors is used to compute the next hidden representation of target node v_i and is defined as

$$\mathbf{H}_{i}^{k+1} = \sum_{v_j \in N(i)\cup\{v_i\}} \alpha_{ij}^{k}\mathbf{H}_{j}^{k}. \tag{29}$$

Since the whole AGNN model consists of several same propagation layer, Kiran et al. [59] propose the following layer-wise propagation rule based on a recursive formulation $\mathbf{H}^{k+1} = \mathbf{P}^{k}\mathbf{H}^{k}$, where $\mathbf{P}^{k} \in \mathbb{R}^{n\times n}$ is the coefficient matrix derived from the k-th propagation layer. Finally, the output layer consists of a linear transformer and an activation function to produce the final representation for each node

$$\mathbf{Z} = f(\mathbf{X}, \mathbf{A}) = \sigma\left(\mathbf{H}^{K}\mathbf{W}^{1}\right), \tag{30}$$

where $\mathbf{W}^1 \in \mathbb{R}^{h\times f}$ is the transformation weight matrix, K denotes the number of attentional propagation layers, and $\sigma(\cdot)$ is implemented by softmax function. Note that $\mathbf{W}^0$, $\mathbf{W}^1$, and $\beta^{(k)}$ are the parameters that need to learn in the training process.

Instead of using all nodes in a graph to compute attention coefficients [60,61], Kiran et al. [59] only used the first-order neighbors similar to the GAT model [57]. The illustration of AGNN is depicted in Fig. 8.

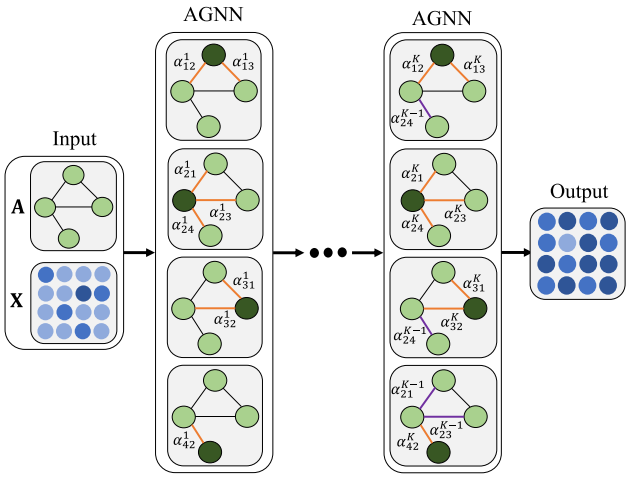

Fig. 8 Schematic depiction of the multi-layer AGNN model. α_{ij}^{k} denotes the importance from node v_j to v_i in the k-th layer. The representation of target node is obtained by combining its neighborhoods with various weights in different layers. The self-connection is not considered

3.3 Autoencoder mechanism

Autoencoder mechanism is one of the most commonly used unsupervised technology to learn a low-dimensional embedding from large unlabeled training data. Moreover, there is existing a large number of node classification data with scarce labeled nodes. Thus, it is essential to utilize this mechanism to learn a higher representation for all nodes.

3.3.1 VGAE

Kipf and Welling [62] firstly employed the variational autoencoder (VAE) [63] to deal with the graph-structured data and proposed an unsupervised learning algorithm variational graph autoencoder (VGAE). The framework of VGAE can be separated into two parts: the inference model (encoder) and the generative model (decoder).

In detail, the inference model is implemented by only using a two-layer GCN model [48] and can be defined as

$$q(\mathbf{Z}|\mathbf{X}, \mathbf{A}) = \prod_{i=1}^{n} q\left(\mathbf{Z}_i|\mathbf{X}, \mathbf{A}\right), \tag{31}$$

where $q\left(\mathbf{Z}_i|\mathbf{X}, \mathbf{A}\right) = \mathcal{N}\left(\mathbf{Z}_i|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)$, $\boldsymbol{\mu}_i \in \mathbb{R}^f$ is the mean vector, $\boldsymbol{\Sigma}_i \in \mathbb{R}^{f\times f}$ denotes the covariance matrix of above Gaussian distribution. $\mathbf{Z}_i$ denotes the stochastic latent variable for the i-th node and is used to construct the latent matrix $\mathbf{Z} \in \mathbb{R}^{n\times f}$. Specifically, $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ are obtained by using the GCN layer which is defined as $\boldsymbol{\mu}_i = \mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X}_i, \mathbf{A})$ and $\log \boldsymbol{\Sigma}_i = \mathrm{GCN}_{\boldsymbol{\Sigma}}(\mathbf{X}_i, \mathbf{A})$, respectively. Then, the definition of this two-layer GCN model is shown as follows:

$$\mathrm{GCN}(\mathbf{X}, \mathbf{A}) = \mathring{\mathbf{A}}\sigma\left(\mathring{\mathbf{A}}\mathbf{X}\mathbf{W}^{0}\right)\mathbf{W}^{1}, \tag{32}$$

where $\mathring{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ denotes the symmetrically normalized adjacency matrix and $\sigma(\cdot)$ is implemented by a ReLU function; $\mathbf{W}^0$ and $\mathbf{W}^1$ are the learnable weight matrices and $\mathbf{W}^0$ is shared between $\mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X}_i, \mathbf{A})$ and $\mathrm{GCN}_{\boldsymbol{\Sigma}}(\mathbf{X}_i, \mathbf{A})$.

In the generative process, Kipf and Welling [62] simply employed an inner product operation between different learned latent variables to implement the decoder and can be formed as:

$$p\left(\mathbf{A}_{ij} = 1|\mathbf{Z}_i, \mathbf{Z}_j\right) = \sigma\left(\mathbf{Z}_i^{T}\mathbf{Z}_j\right). \tag{33}$$

Here, $\sigma(\cdot)$ is implemented by a sigmoid activation function $\mathrm{sigmoid}(x) = 1/\left(1 + e^{-x}\right)$. Finally, the goal of VGAE is to minimize the variational lower bound:

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A}|\mathbf{Z})] - \mathrm{KL}[q(\mathbf{Z}|\mathbf{X},\mathbf{A}) \| p(\mathbf{Z})], \quad (34)$$

with a Gaussian prior distribution

$$p(\mathbf{Z}) = \prod_{i=1}^{n} p(\mathbf{Z}_i) = \prod_{i=1}^{n} \mathcal{N}(\mathbf{Z}_i|\mathbf{0}, \mathbf{I}_n). \quad (35)$$

Here, $\mathbf{0}$ is a zero vector and $\mathrm{KL}[\cdot \parallel \cdot]$ operation denotes the Kullback–Leibler (KL) divergence between two distributions. Note that, Kipf and Welling [62] set $\mathbf{A}_{ij} = 1$ in Eq. (34) when $\mathbf{A}$ is very sparse and used the *renormalization trick* strategy during the training process.

Additionally, Kipf and Welling [62] proposed a non-probabilistic variant of VGAE called graph autoencoder (GAE). In the GAE model, the embeddings of all nodes are computed as same as Eq. (32) and are then used to reconstruct $\mathbf{A} = \sigma\left(\mathbf{Z}\mathbf{Z}^T\right)$.

3.3.2 G2G

In fact, the previous approaches represent each node by a single point vector in a low-dimensional continuous space, and it is difficult to obtain information about the uncertainty of representation. However, diverse nodes may contain different uncertainties. On the other hand, identical uncertainty is insufficient for modeling [64]. As a solution to this problem, Bojchevski et al. [65] proposed a new graph autoencoder model called Graph2Gauss (G2G) to represent each node as a low-dimensional Gaussian distribution which allows capturing the uncertainty of representations.

In detail, G2G uses a deep encoder $f_\theta(\mathbf{X}_i)$ to map nodes to a middle representation $\mathbf{M}_i$ which is then used as an input to obtain the mean vector $\boldsymbol{\mu}_i = \sigma(\mu_\theta(\mathbf{M}_i))$ and covariance matrix $\boldsymbol{\Sigma}_i = \sigma(\Sigma_\theta(\mathbf{M}_i))$ of the Gaussian distribution, respectively. Here, $\boldsymbol{\mu}_i \in \mathbb{R}^f$, $\boldsymbol{\Sigma}_i \in \mathbb{R}^{f\times f}$ with $f \ll n, d$, and $\sigma(\cdot)$ is implemented by ReLU or other nonlinear functions. θ are parameters that need to be trained and shared by all instances. Although $\mu_\theta(\cdot)$ and $\Sigma_\theta(\cdot)$ are usually feed-forward neural networks, it can be CNNs or RNNs model where each node denotes an image or a sequence, respectively. Then, $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ are pass through a Gaussian distribution to obtain the node embedding $\mathbf{H}_i$, which is shown as follows:

$$\mathbf{H}_i = \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \quad (36)$$

In order to utilize the information of graph structure during learning node representation, Bojchevski et al. [65] proposed an unsupervised personalized ranking algorithm. The algorithm is constrained to satisfy the following pairwise constraints

$$\Delta(\mathbf{H}_i, \mathbf{H}_j) < \Delta(\mathbf{H}_i, \mathbf{H}_{j'}) \quad (37)$$

where $\forall v_i \in \mathcal{V}$, $\forall v_j \in N_k(i)$, $\forall v_{j'} \in N_{k'}(i)$, $\forall k < k'$ and $\Delta(\mathbf{H}_i, \mathbf{H}_j)$ denotes the dissimilarity measure between the Gaussian distribution embedding of v_i and v_j. Equation (37) expresses the intuition which the 1-hop neighbors of v_i should be closer to node v_i than the 2-hop neighbors in the embedding space and so on.

Since the dissimilarity measure $\Delta(\mathbf{H}_i, \mathbf{H}_j)$ is defined between the Gaussian distribution of two nodes, it is natural to use the (KL) divergence similar to He et al. [64] and Santos et al. [66]. Specifically, $\Delta(\mathbf{H}_i, \mathbf{H}_j)$ can be defined by an asymmetric KL divergence which is shown as follows:

$$\begin{aligned}\Delta(\mathbf{H}_i, \mathbf{H}_j) &= \mathrm{KL}(\mathcal{N}(i) \parallel \mathcal{N}(j)) \\ &= \frac{1}{2}[tr(\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\Sigma}_j) + \mathbf{P} - \mathbf{L} - log\frac{det(\boldsymbol{\Sigma}_j)}{det(\boldsymbol{\Sigma}_i)}]\end{aligned} \quad (38)$$

where $\mathbf{P} = (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_i^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)$, $tr(\cdot)$ denotes the trace of a matrix, and $det(\cdot)$ denotes the determinant of a matrix.

Since it is difficult to solve Eq. (37), Bojchevski et al. [65] introduced an energy-based method to define the energy between two nodes by using the KL divergence. Then, the objective of G2G is to optimize the following loss function

$$\begin{aligned}\mathcal{L} &= \sum_i \sum_{k<l} \sum_{v_{j_k} \in N_k(i)} \sum_{v_{j_l} \in N_l(i)} ({E_{ij_k}}^2 + \exp^{-E_{ij_l}}) \\ &= \sum_{(i,j_k,j_l)\in\mathcal{D}_t} ({E_{ij_k}}^2 + \exp^{-E_{ij_l}})\end{aligned} \quad (39)$$

where $E_{ij} = \mathrm{KL}(\mathcal{N}(i)||\mathcal{N}(j))$ is the energy between two nodes, and $\mathcal{D}_t = \{(i, j_k, j_l)|\,\mathrm{sp}(i, j_k) < \mathrm{sp}(i, j_l)\}$ is the set of all valid instances. Additionally, Bojchevski et al. [65] proposed an unbiased node-anchored sampling (NaS) strategy to solve the unbalanced problem, which may occur in the original sampling strategy.

3.3.3 DGI

The previous unsupervised learning methods used for generating node embedding are mostly based on the random walk objective. However, these approaches pay too much attention to the localized structure information [67], and the performance relies heavily on the setting of hyperparameter [68]. To solve this problem, Veličković et al. [69] proposed an unsupervised learning method call deep graph infomax (DGI), which is based on the mutual information between the global representation of the entire graph and the patch representation of special input. Specifically, mutual information maximization is introduced into the graph data by DGI.

In detail, Veličković et al. [69] produced a high-level embedding $\mathbf{H}_i$ for each node v_i by using a graph convolutional encoder $f : \mathbb{R}^{n\times d} \times \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times f}$, such that

$\mathbf{H} = f(\mathbf{X}, \mathbf{A})$. Note that Veličković et al. [69] referred $\mathbf{H}_i$ as the patch representation of v_i since this embedding is formed by summarizing a patch of graph information. Veličković et al. [69] introduced a summary vector $\mathbf{s}$ to denote the global information of the entire graph. The summary vector $\mathbf{s}$ is computed by using a readout function $\mathcal{R} : \mathbb{R}^{n\times d} \rightarrow \mathbb{R}^d$, which uses the derived patch representations as input and can be formulated as

$$\mathbf{s} = \mathcal{R}(f(\mathbf{X}, \mathbf{A})) = \mathcal{R}(\mathbf{H}). \tag{40}$$

Then, Veličković et al. [69] assigned a probability score for each patch-summary pair by using a discriminator, $\mathcal{D} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$. Note that this score will then be used in the objective function.

Furthermore, a negative sample strategy is employed to produce the product of marginals (negative samples). Specifically, an explicit (stochastic) corruption function, $\mathcal{C} : \mathbb{R}^{n\times d} \times \mathbb{R}^{n\times n} \rightarrow \mathbb{R}^{h\times d} \times \mathbb{R}^{h\times h}$, is used to obtain the alternative graph derived from the original graph which is shown as follows:

$$\breve{\mathcal{G}} = (\breve{\mathbf{X}}, \breve{\mathbf{A}}) = \mathcal{C}(\mathbf{X}, \mathbf{A}), \tag{41}$$

where h denotes the number of nodes in the alternative graph $\breve{\mathcal{G}}$. Note that Eq. (41) is only used for the situation where the dataset only contains one single graph. For the multi-graphs situation, some other graphs of the dataset will be used as $\breve{\mathcal{G}}$. Thus, the negative samples can be derived from the alternative graph.

Finally, in order to maximize the mutual information between $\mathbf{H}_i$ and $\mathbf{s}$, Veličković et al. [69] used a noise-contrastive type objective with a standard binary cross-entropy loss between the samples from the joint and the product of marginals and can be formed as

$$\mathcal{L} = \frac{1}{n+h}\left(\mathcal{L}_{pos} + \mathcal{L}_{neg}\right), \tag{42}$$

with

$$\mathcal{L}_{pos} = \sum_{i=1}^{n} \mathbb{E}_{(\mathbf{X},\mathbf{A})}\left[\log \mathcal{D}(\mathbf{H}_i, \mathbf{s})\right],$$

and

$$\mathcal{L}_{neg} = \sum_{j=1}^{h} \mathbb{E}_{(\breve{\mathbf{X}},\breve{\mathbf{A}})}\left[\log\left(1 - \mathcal{D}\left(\breve{\mathbf{H}}_j, \mathbf{s}\right)\right)\right].$$

Here, the calculation of Eq. (42) is based on the Jensen–Shannon divergence between the positive examples and the negative examples. The DGI model is illustrated in Fig. 9.

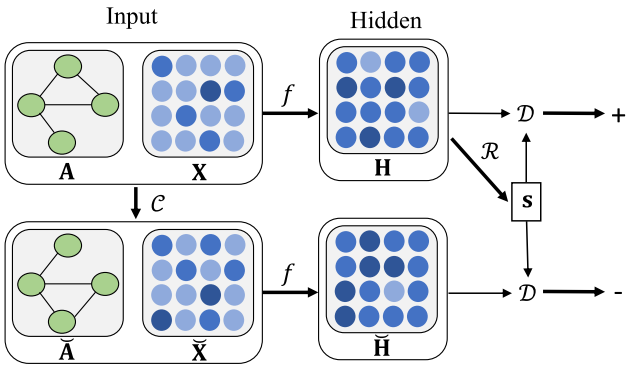

Fig. 9 A high-level overview of DGI. f represents a graph convolutional encoder. $\mathcal{R}$, $\mathcal{D}$, $\mathcal{C}$ denote the readout function, discriminator and corruption function, respectively. The hidden embedding $\mathbf{H}$ is used as the input of $\mathcal{R}$ to obtain the summary vector $\mathbf{s}$

4 Experimental analysis

In this section, we compare the aforementioned graph neural networks on several popular node classification datasets. Firstly, we describe the statistic information of datasets and parameter settings in detail. Then, four commonly used evaluation metrics are introduced to evaluate the performance of various algorithms. The classification results of different methods are provided at the end.

4.1 Datasets

We conduct our experiments on several applications, including citation networks, co-author networks, Amazon networks, protein-protein interaction (PPI) networks. The detailed descriptions of each dataset are introduced in the following, and the statistic information of all datasets is summarized in Table 2.

4.1.1 Transductive learning

Transductive learning means that the trained models miss the ability to generalize to unseen nodes. In this category, several standard benchmark datasets, including five citation networks, one co-author network and two Amazon networks, are used to compare. For all of these datasets, the transductive experimental setup is similar to Yang et al. [70].

In detail, Cora, Citeseer and Pubmed are firstly introduced by Sen et al. [41] and DBLP is proposed by Pan et al. [71]. The Cora-ML dataset is similar to Cora and was produced by Bojchevski et al. [65]. In all of these citation datasets, each node represents a scientific paper and edges correspond to citations. Note that the citation networks are represented as undirected graphs according to [70]. Node features are bag-of-words encoded documents. The class labels represent the research domains to which each document belongs, and each

Table 2 Dataset statistics. The statistic information of datasets used in our experiments

Dataset	Type	Task	Nodes	Edges	Features	Classes	Train/test nodes
Cora	Citation network	Transductive	2708	5278	1433	7	140/1000
Citeseer	Citation network	Transductive	3327	4552	3703	6	120/1000
Pubmed	Citation network	Transductive	19,717	44,324	500	3	60/1000
DBLP	Citation network	Transductive	17,716	52,867	1639	4	80/1000
Cora-ML	Citation network	Transductive	2995	8158	2879	7	140/1000
CS	Co-author network	Transductive	18,333	81,894	6805	15	300/1000
Photo	Amazon network	Transductive	7650	119,081	745	8	160/1000
Computers	Amazon network	Transductive	13,752	245,861	767	10	200/1000
PPI	PPI network	Inductive	56,944 (24 graphs)	818,716	50	121 (multi-labels)	44,906/5524 (20/2 graphs)

node has a unique tag. For co-author graph CS, each node represents a researcher connected by an edge if they co-author a paper. Node features correspond to paper keywords for each author's papers. The labels of nodes denote the most active fields of study for each author. In Amazon networks, Computers and Photo are segments of the Amazon co-purchase graph [72], where vertices denote goods and node features correspond to elements of a bag-of-words representation of a review. If two products are frequently bought together, there is an edge between them. The class labels represent the category of each product.

In order to obtain the performance of each model in all transductive datasets, only 20 nodes for each class are used to train, and the other 1000 nodes are used in the test phase. Note that all of the node features are utilized in the unsupervised learning methods and all graphs are undirected.

4.1.2 Inductive learning

Contrary to transductive learning, an algorithm is inductive means that it can be generalized to unseen graphs or nodes. In our experiments, a popular PPI network benchmark dataset is used in this setting.

In detail, the complete PPI dataset is provided by Zitnik and Leskovec [73] and then was preprocessed again by Hamilton et al. [50]. The version of PPI used in our experiments is the latter one. In this dataset, each graph corresponds to different human tissues, and nodes represent various protein functions. The features of each node are derived from positional gene sets, motif gene sets and immunological signatures. Each label represents a gene ontology set collected from the Molecular Signatures Database [43], and each node can hold several labels simultaneously. The dataset contains 20 graphs used in the training stage and two completely unobserved graphs for testing.

4.2 Experimental setup

Since most parameters of all algorithms are set according to the original papers, the validation procedure is stripped in our experiments. For all models, they have some identical settings introduced in the following. The maximum number of training epochs is 2000, and the training stage would be early stopped when the loss is not reducing in one consecutive hundred epochs. We repeatedly conduct each experiment five times and provide the values of mean and variance across all repeats. The parameters of all methods are optimized by using Adam [74]. For supervised or semi-supervised learning models, the objective of model training is to minimize the cross-entropy loss. On the other hand, G2G and DGI both have a customize loss function. For instance, cross-entropy and KL divergence are both used in VGAE. Similar to transductive datasets, the multi-label classification of the

PPI dataset is regarded as a multi-class classification problem.

For those models in which node classification is not considered in the original paper, the initial configurations are referring to algorithms similar to their architecture. For instance, the parameter initialization of ChebNet is derived from GCN's settings, and VGAE is similar to G2G. Note that the sampling technology of GraphSAGE introduced in Sect. 3 is not used in our experiments, and the mean aggregator is used to propagate information. After the embedding matrix is obtained, a linear transform function is trained to gain the probability distribution of classes for unsupervised learning methods, including VGAE, G2G and DGI.

The major software and hardware devices used in our experiments are listed in the following. Based on the deep learning framework PyTorch[1], an open source graph representation learning tool called PyTorch Geometric[2] (PyG) was used in our experiments. Specifically, the latest released version of PyG, 1.3.0, is used to implement all baseline methods except the G2G model. G2G is implemented by using the code provided by Bojchevski et al. [65] based on the TensorFlow framework[3]. Note that the train data and test data used in G2G are proportional to the full dataset according to a rule in which the final size of training data is approximate to other baseline methods. In order to accelerate the training process, one graphics processing unit (GPU) card with 16 GB memory is used in all experiments. The implementations of all algorithms are available at https://github.com/Xiaoshunxin/GNN-Survey.

4.3 Evaluation metrics

In order to compare the performance of diverse algorithms, several commonly used evaluation metrics of the classification task, including macro-precision (macro-P), macro-recall (macro-R), macro-F1 and micro-F1, are used in our experiments.

Firstly, the true positive, false positive, true negative and false negative of i-th class are denoted as TP_i, FP_i, TN_i and FN_i, respectively. Then, the definitions of these metrics are introduced in the following.

- In order to obtain the macro-precision of multi-class classification, the precision of each class should be calculated at first and can be formed as $P_i = TP_i/(TP_i + FP_i)$. Then, macro-P is obtained by averaging all the precisions, which is shown as follow

$$\text{macro-P} = \frac{1}{n}\sum_{i=1}^{n} P_i, \tag{43}$$

where n denotes the number of classes.
- Similar, the value of macro-R can be obtained by averaging all the recalls

$$\text{macro-R} = \frac{1}{n}\sum_{i=1}^{n} R_i, \tag{44}$$

where $R_i = TP_i/(TP_i + FN_i)$ denotes the recall of the i-th class.
- In order to balance macro-P and macro-R, Macro-F1 is introduced and can be formed as

$$\text{macro-F1} = \frac{2 \times \text{macro-P} \times \text{macro-R}}{\text{macro-P} + \text{macro-R}}. \tag{45}$$

- Since macro-F1 does not consider the sample size of each class, it is suitable to use micro-F1 when the distribution of classes is unbalanced. The definition of micro-F1 is formed as

$$\text{micro-F1} = \frac{2 \times \text{micro-P} \times \text{micro-R}}{\text{micro-P} + \text{micro-R}}, \tag{46}$$

with

$$\text{micro-P} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n}(TP_i + FP_i)},$$

and

$$\text{micro-R} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n}(TP_i + FN_i)}.$$

4.4 Experimental results

In this section, the compared algorithms are tested on nine publicly available datasets with several evaluation metrics mentioned above. The macro-P, macro-R, macro-F1 and micro-F1 of all algorithms on all datasets are listed in Table 3.

The blank cell described by one horizontal line represents that this algorithm is essential unsuitable to current dataset except for G2G and DGI. For instance, transductive learning algorithms including GCN, VGAE, ChebNet, APPNP and SGC cannot be used on the PPI dataset because this dataset is inductive. However, the performance of G2G and DGI on the PPI dataset is not obtained because the setting of inductive learning is not provided in the source code. In addition, the SGC model and GAT model have both obtained the best

[1] https://github.com/pytorch/pytorch.

[2] https://pytorch-geometric.readthedocs.io/en/latest/.

[3] https://github.com/tensorflow/tensorflow.

Table 3 The performance (mean% and standard deviation%) of diverse GNNs algorithms on different datasets. The best results are marked in bold (The higher the better)

Datasets \ Methods		ChebNet	GCN	GraphSAGE	APPNP	SGC	GAT	AGNN	VGAE	G2G	DGI
Cora	macro-P	79.9 (0.8)	79.9 (0.3)	75.1 (1.5)	69.6 (2.2)	78.7 (0.0)	**80.4 (0.6)**	78.3 (1.6)	74.9 (1.2)	79.3 (0.1)	78.9 (0.6)
	macro-R	**83.2 (0.5)**	82.4 (0.3)	78.6 (0.5)	69.7 (0.8)	81.0 (0.0)	83.0 (0.5)	79.5 (1.2)	79.0 (1.0)	75.4 (0.3)	81.6 (0.7)
	macro-F1	81.2 (0.7)	80.9 (0.3)	75.7 (1.3)	61.4 (0.6)	79.7 (0.0)	**81.3 (0.5)**	78.3 (1.3)	76.0 (1.3)	76.4 (0.2)	79.9 (0.7)
	micro-F1	82.0 (0.6)	81.7 (0.3)	75.9 (1.6)	56.4 (0.4)	80.7 (0.0)	**82.2 (0.6)**	79.1 (1.3)	76.7 (1.3)	79.4 (0.2)	81.1 (0.9)
CiteSeer	macro-P	66.5 (0.8)	67.6 (0.2)	61.2 (1.6)	19.2 (0.6)	**68.9 (0.1)**	67.9 (0.7)	67.6 (0.7)	61.0 (0.4)	64.1 (0.5)	67.7 (0.5)
	macro-R	67.0 (0.8)	68.0 (0.2)	59.6 (0.8)	37.4 (0.5)	**69.5 (0.1)**	68.3 (1.1)	68.0 (0.7)	59.9 (0.6)	63.0 (0.3)	68.1 (0.5)
	macro-F1	66.3 (0.8)	67.4 (0.2)	59.0 (1.1)	24.9 (0.5)	**68.9 (0.1)**	67.6 (1.1)	67.4 (0.8)	59.8 (0.7)	62.2 (0.5)	67.6 (0.5)
	micro-F1	69.0 (0.6)	70.4 (0.2)	61.9 (1.0)	38.8 (0.3)	**71.5 (0.0)**	70.5 (1.3)	70.1 (0.9)	64.1 (1.3)	68.6 (0.4)	71.2 (0.5)
PubMed	macro-P	75.9 (1.2)	78.3 (0.6)	75.0 (1.5)	76.8 (0.8)	78.4 (0.0)	76.8 (0.3)	78.2 (0.3)	77.3 (0.7)	**82.1 (0.3)**	77.1 (0.3)
	macro-R	75.6 (1.0)	79.1 (0.6)	75.9 (1.0)	77.2 (0.5)	78.7 (0.0)	77.3 (0.3)	79.5 (0.4)	79.2 (1.0)	**80.3 (0.8)**	79.4 (0.3)
	macro-F1	75.6 (0.9)	78.5 (0.2)	75.1 (1.1)	76.6 (0.6)	78.5 (0.0)	76.8 (0.3)	78.5 (0.4)	78.1 (0.7)	**81.0 (0.6)**	78.0 (0.3)
	micro-F1	75.9 (1.1)	78.9 (0.2)	75.7 (1.2)	77.0 (0.6)	79.0 (0.0)	77.4 (0.4)	79.1 (0.5)	78.8 (1.0)	**81.9 (0.6)**	77.9 (0.3)
DBLP	macro-P	61.9 (0.7)	60.6 (0.8)	54.8 (3.1)	65.7 (0.8)	56.1 (0.2)	63.4 (1.7)	59.3 (3.7)	52.2 (3.1)	**74.6 (0.5)**	66.5 (1.6)
	macro-R	64.6 (2.0)	64.4 (1.1)	61.4 (3.9)	**72.5 (0.9)**	56.0 (0.4)	69.9 (1.0)	62.9 (2.9)	55.5 (3.0)	68.5 (1.9)	72.1 (1.9)
	macro-F1	58.0 (1.5)	56.9 (2.1)	52.4 (5.8)	65.5 (1.3)	52.0 (0.3)	63.8 (1.6)	54.8 (4.5)	46.7 (4.5)	**70.7 (1.2)**	66.4 (1.7)
	micro-F1	57.2 (2.3)	57.1 (2.9)	54.1 (6.6)	67.5 (1.6)	53.1 (0.4)	66.5 (1.3)	54.5 (4.5)	46.6 (5.1)	**77.9 (0.5)**	68.1 (2.0)
Cora-ML	macro-P	67.5 (1.3)	67.5 (1.0)	64.5 (2.3)	69.0 (0.5)	65.5 (0.3)	68.6 (1.2)	69.1 (0.6)	67.3 (1.0)	**80.6 (0.6)**	69.7 (0.3)
	macro-R	72.7 (0.9)	71.9 (0.8)	68.1 (1.4)	74.3 (0.3)	71.1 (0.2)	72.1 (0.8)	72.8 (0.5)	68.0 (1.7)	**75.6 (0.5)**	73.9 (0.6)
	macro-F1	68.1 (1.3)	68.5 (1.1)	65.0 (1.8)	69.6 (0.5)	66.9 (0.3)	69.2 (0.0)	69.5 (0.7)	65.6 (1.4)	**76.2 (0.4)**	70.0 (0.4)
	micro-F1	67.6 (1.5)	68.8 (1.3)	65.8 (1.7)	69.7 (0.5)	67.9 (0.2)	69.5 (1.0)	69.6 (0.8)	64.4 (1.8)	**79.6 (0.2)**	69.4 (0.3)
CS	macro-P	70.4 (2.5)	87.5 (0.9)	80.5 (1.4)	88.0 (0.4)	85.6 (0.0)	87.4 (0.6)	**88.3 (1.3)**	77.4 (1.5)	62.6 (1.4)	87.0 (0.6)
	macro-R	62.2 (4.6)	**88.0 (0.7)**	84.1 (0.4)	87.8 (0.3)	85.6 (0.3)	87.0 (0.6)	87.4 (1.4)	83.6 (0.6)	60.3 (1.3)	87.9 (0.4)
	macro-F1	53.6 (5.0)	87.3 (0.8)	81.4 (1.0)	**87.5 (0.4)**	85.0 (0.4)	86.8 (0.6)	87.3 (1.6)	78.7 (1.4)	60.7 (1.4)	87.0 (0.5)
	micro-F1	58.4 (6.7)	**89.3 (0.7)**	84.9 (0.4)	89.2 (0.4)	88.4 (0.3)	88.4 (0.4)	89.0 (1.6)	81.2 (1.3)	68.8 (1.1)	88.7 (0.3)
Photo	macro-P	86.2 (1.5)	86.1 (1.1)	86.4 (1.6)	85.6 (1.4)	83.7 (0.2)	**87.0 (1.5)**	81.0 (4.4)	85.3 (0.3)	51.2 (0.9)	83.9 (0.8)
	macro-R	89.6 (1.6)	90.9 (0.6)	89.8 (2.9)	90.1 (1.0)	89.6 (0.2)	**91.0 (1.1)**	82.2 (3.8)	90.7 (0.3)	50.4 (0.8)	89.4 (0.5)
	macro-F1	87.4 (1.6)	87.8 (0.9)	86.6 (4.1)	86.8 (1.2)	85.7 (0.1)	**88.2 (1.8)**	80.6 (4.1)	86.8 (0.3)	49.6 (0.8)	85.5 (0.7)
	micro-F1	88.5 (1.2)	89.0 (0.7)	87.3 (4.1)	87.7 (1.5)	87.4 (0.1)	**89.5 (1.1)**	84.6 (2.0)	88.0 (0.0)	66.3 (0.9)	86.6 (1.0)
Computers	macro-P	71.6 (1.6)	78.1 (1.3)	77.5 (1.0)	79.5 (1.5)	79.0 (0.6)	**79.8 (1.3)**	72.6 (4.9)	75.3 (1.6)	45.2 (1.5)	76.6 (1.1)
	macro-R	80.0 (1.3)	86.8 (1.7)	87.8 (1.0)	86.4 (3.4)	**88.2 (0.2)**	88.2 (0.6)	74.2 (8.3)	86.3 (0.8)	46.8 (0.8)	85.6 (1.0)
	macro-F1	75.8 (1.4)	81.1 (1.5)	80.7 (0.5)	80.9 (2.2)	82.2 (0.4)	**82.7 (1.1)**	69.5 (7.1)	79.3 (1.5)	45.5 (1.1)	79.7 (1.1)
	micro-F1	76.8 (1.8)	81.9 (1.3)	81.1 (0.8)	82.1 (1.7)	82.8 (0.1)	**83.5 (1.1)**	68.0 (7.9)	79.6 (0.5)	67.8 (0.4)	81.2 (0.9)
PPI	macro-P	–	–	72.8 (1.6)	–	–	**99.0 (0.0)**	72.8 (1.9)	–	–	–
	macro-R	–	–	63.8 (1.1)	–	–	**98.4 (0.1)**	59.9 (1.4)	–	–	–
	macro-F1	–	–	66.1 (0.9)	–	–	**98.7 (0.1)**	61.6 (0.9)	–	–	–
	micro-F1	–	–	72.4 (0.6)	–	–	**98.8 (0.1)**	70.6 (0.5)	–	–	–

performance on the Computers dataset in macro-recall metric. We regard SGC as the best model because of the smaller variance.

Furthermore, several observations are derived from the results shown in Table 3. From the perspective of the algorithm, some algorithms are suitable for a special graph. For instance, GAT achieves the best performance on two transductive datasets (Cora and Photo) and one inductive benchmark (PPI) with all evaluation metrics. This observation can reflect the importance of attention mechanism in both transductive and inductive learning. SGC model obtains the minimum variances on all transductive learning datasets with all evaluation metrics. This phenomenon may reflect that the simpler a model is, the more stable it is. In addition, SGC achieves comparable performance on several datasets and obtains the best result on the CiteSeer benchmark with all evaluation metrics. It demonstrates that a simple method, which only has a few layers, may have comparable performance to complex models. G2G algorithm also achieves the best performance on three datasets (PubMed, DBLP and Cora-ML) in four criterions, except for the Macro-recall result on the DBLP dataset. This phenomenon demonstrates that not only the supervised information (e.g., node labels) is critical, but also the extensive unsupervised information (e.g., the information of graph structure) is all important for the node classification task. On the other hand, observations would be derived from the perspective of datasets. AGNN usually has a big variance on large datasets such as DBLP, Photo and Computers. In addition, the performance of different algorithms is greatly different on the same dataset. For instance, the gap between the best method and the worst network on the CiteSeer dataset is close to forty percent. It demonstrates that different algorithms may be suitable for different tasks.

The runtimes of each training epoch of different GNNs algorithms on several datasets are illustrated in Fig. 10. For those models which training embeddings in an unsupervised way, only the time cost of training hidden representation for each node is considered in our experiments. We also have some observations from this figure. The training time of unsupervised learning algorithms including VGAE, G2G and DGI is longer than those semi-supervised or supervised methods on most datasets. ChebNet has a larger time cost on CS because this model is sensitive to the size of nodes set and edge set, as well as the dimension of node features. The SGC model spends very little time on all datasets because there is only one linear transformation in the training stage.

5 Challenges and future directions

In this section, several challenges of the development of GNNs would be introduced based on the experimental results mentioned above. Then, some potential future directions are provided for interested researchers.

Based on the observations derived from the experimental results, we can find many existing problems in previous research works.

- In transductive learning, several aforementioned graph analysis algorithms are difficultly scaled to large graphs in real-world applications such as community detection, which usually contains millions of nodes and edges.
- Most of the existing graph neural networks are essential transductive learning algorithms. This means that they cannot handle nodes or graphs that never appear and are unsuitable for many applications.
- Several deep learning paradigms used for graph analysis are supervised or semi-supervised learning. These methods often require lots of supervised information in the training stage to obtain a good performance. However, most of the existing graph data are unsupervised and it is difficult to apply the supervised algorithms to these unlabeled data.
- The applications of node classification are mainly focused on those data in which there are existing real-world relationships between samples. However, it is a possible way to apply GNNs in different kinds of data via node classification.
- Stacking several layers of graph convolution may cause over-smoothing that the learned representations of all nodes tend to be the same. As a result, the performance of graph convolution models would decrease dramatically.

Then, several potential future directions are introduced based on the challenges mentioned above.

- The ability to deal with a large graph is important in many scenes. From the perspective of computational resources, the possible solution is to design a parallelized algorithm that can run with multiple GPU cards. On the other hand, many effective sampling techniques would be introduced to reduce the scale of large graphs.
- Generalizing to unseen nodes or graphs has many benefits. For instance, the model can be trained only in some parts of the whole graph and then used to predict other parts. One of the essential reasons for losing the inductive ability is that they usually apply the information aggregation on the propagation matrix. As a result, we suggest that the information aggregation should be partial instead of global.
- There is a lot of unsupervised information in graph data. Using this information can produce a meaningful representation for nodes or graphs. We can use unsupervised deep learning technologies such as autoencoder mechanism to achieve this objective.

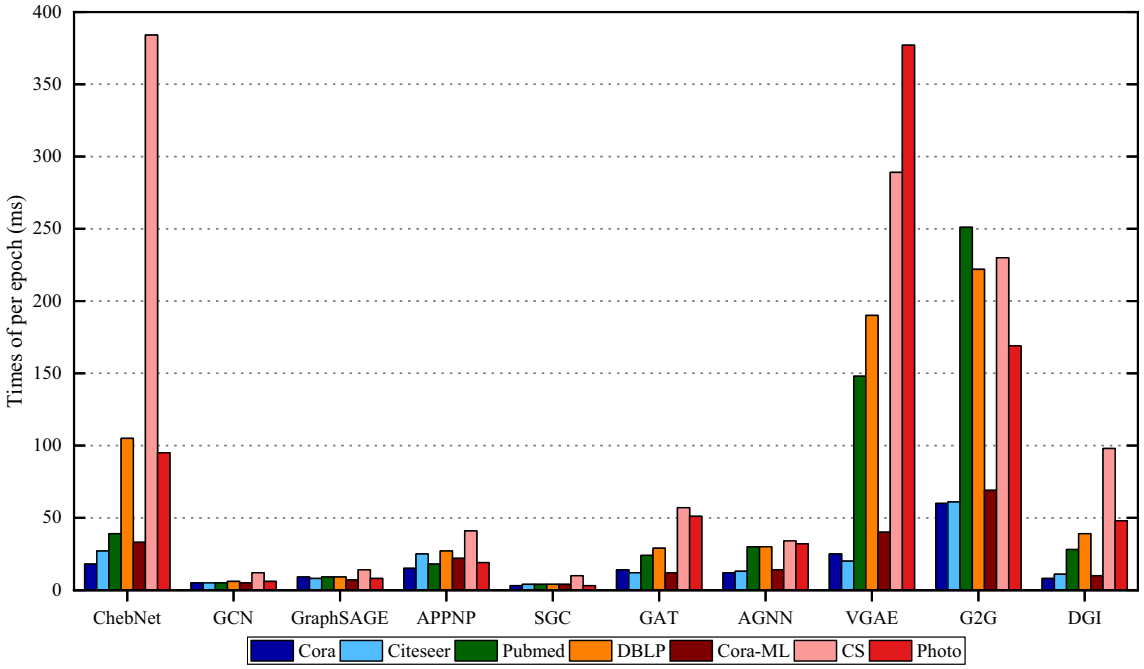

Fig. 10 Runtime of training stage. The time cost of each training epoch of diverse GNNs algorithms on several transductive learning datasets

– For common classification or clustering tasks, heuristic methods can be used to model the real relationship between samples. Then, these tasks can be converted to node classification and be learned by various GNNs models.
– To solve the over-smoothing problem, one can introduce external information to reduce the distortion caused by deep training. For example, the relationships between nodes can be used as supervised information to guide the training process.

Finally, we believe that the designed GNNs model would be more accurate and more quickly by further studying these directions.

6 Conclusion

In this paper, we introduced an overview of graph neural networks and provided a comparison among them in node classification tasks. According to the major learning paradigms, these graph analysis algorithms were divided into three categories: convolutional mechanism, attention mechanism and autoencoder mechanism. A comprehensive introduction of several popular methods for each category was provided. Furthermore, extensive comparative experiments were conducted on nine benchmark datasets to compare the performance of node classification. Finally, a number of challenges are introduced based on the experimental results and several potential future directions are provided to interested researchers. In the future work, we will explore more efficient graph neural networks for node classification based on these future directions.

Acknowledgements This work is partly supported by the National Natural Science Foundation of China under Grant Nos. 61502104 and 61672159.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Comput. **1**(4), 541–551 (1989)
2. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Proceedings of the 26th Conference on Neural Information Processing Systems, pp. 1097–1105 (2012)
3. Wan, W., Zhong, Y., Li, T., Chen, J.: Rethinking feature distribution for loss functions in image classification. In: Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition, pp. 9117–9126 (2018)
4. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3431–3440 (2015)
5. Papadomanolaki, M., Vakalopoulou, M., Karantzalos, K.: A novel object-based deep learning framework for semantic segmentation

of very high-resolution remote sensing data: Comparison with convolutional and fully convolutional networks. Remote Sens. **11**(6), 684 (2019)
6. Chen, L., Zhang, H., Xiao, J., Nie, L., Shao, J., Liu, W., Chua, T.S.: Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning. In: Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition, pp. 5659–5667 (2017)
7. Aneja, J., Deshpande, A., Schwing, A.G.: Convolutional image captioning. In: Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition, pp. 5561–5570 (2018)
8. Elman, J.L.: Finding structure in time. Cogn. Sci. **14**(2), 179–211 (1990)
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
10. Tang, D., Qin, B., Liu, T.: Document modeling with gated recurrent neural network for sentiment classification. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 1422–1432 (2015)
11. Ma, Y., Peng, H., Cambria, E.: Targeted aspect-based sentiment analysis via embedding commonsense knowledge into an attentive lstm. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, pp. 5876–5883 (2018)
12. Liu, S., Yang, N., Li, M., Zhou, M.: A recursive recurrent neural network for statistical machine translation. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pp. 1491–1500 (2014)
13. Su, J., Wu, S., Xiong, D., Lu, Y., Han, X., Zhang, B.: Variational recurrent neural machine translation. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, pp. 5488–5495 (2018)
14. Xiong, C., Merity, S., Socher, R.: Dynamic memory networks for visual and textual question answering. In: Proceedings of the 33nd International Conference on Machine Learning, pp. 2397–2406 (2016)
15. Lin, Y., Ji, H., Liu, Z., Sun, M.: Denoising distantly supervised open-domain question answering. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, pp. 1736–1745 (2018)
16. Karabatak, M., Ince, M.C.: An expert system for detection of breast cancer based on association rules and neural network. Expert Syst. Appl. **36**(2), 3465–3469 (2009)
17. Cireşan, D.C., Giusti, A., Gambardella, L.M., Schmidhuber, J.: Mitosis detection in breast cancer histology images with deep neural networks. In: Proceedings of the 16th International Conference on Medical Image Computing and Computer-assisted Intervention, pp. 11–418 (2013)
18. Panakkat, A., Adeli, H.: Neural network models for earthquake magnitude prediction using multiple seismicity indicators. Int. J. Neural Syst. **17**(1), 13–33 (2007)
19. Adeli, H., Panakkat, A.: A probabilistic neural network for earthquake magnitude prediction. Neural Netw. **22**(7), 1018–1024 (2009)
20. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature. **529**(7587), 484 (2016)
21. Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., Alsaadi, F.E.: A survey of deep neural network architectures and their applications. Neurocomputing. **234**, 11–26 (2017)
22. Zhang, Q., Yang, L.T., Chen, Z., Li, P.: A survey on deep learning for big data. Inf. Fusion. **42**, 146–157 (2018)
23. Bullmore, E., Sporns, O.: Complex brain networks: graph theoretical analysis of structural and functional systems. Nat. Rev. Neurosci. **10**(3), 186 (2009)
24. Yang, J., Leskovec, J.: Community-affiliation graph model for overlapping network community detection. In: Proceedings of 12th IEEE International Conference on Data Mining, pp. 1170–1175 (2012)
25. Bhatia, V., Rani, R.: A distributed overlapping community detection model for large graphs using autoencoder. Futur. Gener. Comp. Syst. **94**, 16–26 (2019)
26. Huang, W., Song, G., Hong, H., Xie, K.: Deep architecture for traffic flow prediction: Deep belief networks with multitask learning. IEEE Trans. Intell. Transp. Syst. **15**(5), 2191–2201 (2014)
27. Qi, H.: Graphical solution for arterial road traffic flow model considering spillover. IEEE Access. **6**, 6755–6764 (2017)
28. Ji, G., Liu, K., He, S., Zhao, J.: Knowledge graph completion with adaptive sparse transfer matrix. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence, pp. 985–991 (2016)
29. Hamaguchi, T., Oiwa, H., Shimbo, M., Matsumoto, Y.: Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, pp. 1802–1808 (2017)
30. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. In: Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, pp. 729–734 (2005)
31. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Netw. **20**(1), 61–80 (2009)
32. Duvenaud, D.K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., Adams, R.P.: Convolutional networks on graphs for learning molecular fingerprints. In: Proceedings of the 29th Conference on Neural Information Processing Systems, pp. 2224–2232 (2015)
33. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.S.: Gated graph sequence neural networks. In: Proceedings of the 4th International Conference on Learning Representations (2016)
34. Atwood, J., Towsley, D.: Diffusion-convolutional neural networks. In: Proceedings of the 30th Conference on Neural Information Processing Systems, pp. 1993–2001 (2016)
35. Li, Q., Han, Z., Wu, X.M.: Deeper insights into graph convolutional networks for semi-supervised learning. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, pp. 3538–3545 (2018)
36. Zhang, M., Chen, Y.: Link prediction based on graph neural networks. In: Proceedings of the 32nd Conference on Neural Information Processing Systems, pp. 5165–5175 (2018)
37. Xu, X., Liu, C., Feng, Q., Yin, H., Song, L., Song, D.: Neural network-based graph embedding for cross-platform binary code similarity detection. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 363–376 (2017)
38. Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs. In: Proceedings of the 33rd International Conference on Machine Learning, pp. 2014–2023 (2016)
39. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, pp. 4438–4445 (2018)
40. Kazienko, P., Kajdanowicz, T.: Label-dependent node classification in the network. Neurocomputing. **75**(1), 199–209 (2012)
41. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI Mag. **29**(3), 93–93 (2008)
42. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proceedings of the 24th International Conference on World Wide Web, pp. 1067–1077 (2015)
43. Subramanian, A., Tamayo, P., Mootha, V.K., Mukherjee, S., Ebert, B.L., Gillette, M.A., Paulovich, A., Pomeroy, S.L., Golub, T.R., Lander, E.S., et al.: Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. Proc. Natl. Acad. Sci USA **102**(43), 15545–15550 (2005)

44. Xu, J., Li, Y.: Discovering disease-genes by topological features in human protein-protein interaction network. Bioinformatics. **22**(22), 2800–2805 (2006)
45. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Proceedings of the 30th Conference on Neural Information Processing Systems, pp. 3844–3852 (2016)
46. Spielman, D.A.: Spectral graph theory and its applications. In: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science, pp. 29–38. IEEE (2007)
47. Dhillon, I.S., Guan, Y., Kulis, B.: Weighted graph cuts without eigenvectors a multilevel approach. IEEE Trans. Pattern Anal. Mach. Intell. **29**(11), 1944–1957 (2007)
48. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: Proceedings of the 4th International Conference on Learning Representations (2016)
49. Hammond, D.K., Vandergheynst, P., Gribonval, R.: Wavelets on graphs via spectral graph theory. Appl. Comput. Harmon. Anal. **30**(2), 129–150 (2011)
50. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Proceedings of the 31st Conference on Neural Information Processing Systems, pp. 1024–1034 (2017)
51. Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. In: Proceedings of the 35th International Conference on Machine Learning, pp. 5449–5458 (2018)
52. Klicpera, J., Bojchevski, A., Günnemann, S.: Predict then propagate: Graph neural networks meet personalized pagerank. In: Proceedings of the 7th International Conference on Learning Representations (2018)
53. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Tech. rep, Stanford InfoLab (1999)
54. Haveliwala, T.H.: Topic-sensitive pagerank. In: Proceedings of the 11st International Conference on World Wide Web, pp. 517–526. ACM (2002)
55. Wu, F., Zhang, T., Souza, A.H.J., Fifty, C., Yu, T., Weinberger, K.Q.: Simplifying graph convolutional networks. In: Proceedings of the 36th International Conference on Machine Learning, pp. 6861–6871 (2019)
56. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of the 19th International Conference on Computational Statistics, pp. 177–186 (2010)
57. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. In: Proceedings of the 5th International Conference on Learning Representations (2017)
58. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Proceedings of the 31st Conference on Neural Information Processing Systems, pp. 5998–6008 (2017)
59. Thekumparampil, K.K., Wang, C., Oh, S., Li, L.J.: Attention-based graph neural network for semi-supervised learning (2018). arXiv:1803.03735
60. Duan, Y., Andrychowicz, M., Stadie, B., Ho, O.J., Schneider, J., Sutskever, I., Abbeel, P., Zaremba, W.: One-shot imitation learning. In: Proceedings of the 31st Conference on Neural Information Processing Systems, pp. 1087–1098 (2017)
61. Hoshen, Y.: Vain: Attentional multi-agent predictive modeling. In: Proceedings of the 31st Conference on Neural Information Processing Systems, pp. 2701–2711 (2017)
62. Kipf, T.N., Welling, M.: Variational graph auto-encoders (2016). arXiv:1611.07308
63. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. In: Proceedings of the 2nd International Conference on Learning Representations (2013)
64. He, S., Liu, K., Ji, G., Zhao, J.: Learning to represent knowledge graphs with gaussian embedding. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pp. 623–632 (2015)
65. Bojchevski, A., Günnemann, S.: Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In: Proceedings of the 5th International Conference on Learning Representations (2017)
66. Dos Santos, L., Piwowarski, B., Gallinari, P.: Multilabel classification on heterogeneous graphs with gaussian embeddings. In: Proceedings of the 2016 Joint European Conference on Machine Learning and knowledge Discovery in Databases, pp. 606–622 (2016)
67. Ribeiro, L.F., Saverese, P.H., Figueiredo, D.R.: Struc2vec: Learning node representations from structural identity. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 385–394 (2017)
68. Grover, A., Leskovec, J.: Node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 855–864 (2016)
69. Veličković, P., Fedus, W., Hamilton, W.L., Liò, P., Bengio, Y., Hjelm, R.D.: Deep graph infomax. In: Proceedings of the 6th International Conference on Learning Representations (2018)
70. Yang, Z., Cohen, W.W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings. In: Proceedings of the 33rd International Conference on Machine Learning, pp. 40–48 (2016)
71. Pan, S., Wu, J., Zhu, X., Zhang, C., Wang, Y.: Tri-party deep network representation. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence, pp. 1895–1901 (2016)
72. McAuley, J., Targett, C., Shi, Q., Van Den Hengel, A.: Image-based recommendations on styles and substitutes. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 43–52 (2015)
73. Zitnik, M., Leskovec, J.: Predicting multicellular function through multi-layer tissue networks. Bioinformatics. **33**(14), i190–i198 (2017)
74. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Proceedings of the 4th International Conference on Learning Representations (2014)

Shunxin Xiao received the B.S. degree in software engineering from Guilin University of Electronic Technology, Guilin, China, in 2016, and the M.S. degree in computer software and theory from Fuzhou University, Fuzhou, China, in 2019, where he is currently pursuing the Ph.D. degree with the College of Mathematics and Computer Science. His research interests include machine learning, graph neural networks, and computer vision.

Shiping Wang received the Ph.D. degree from the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China, in 2014. He was a Research Fellow with Nanyang Technological University from August 2015 to August 2016. He is currently a Full Professor and Qishan Scholar with the College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China. His research interests include machine learning, data mining, computer vision, optimization theory, and granular computing.

Yuanfei Dai received his Ph.D. degree in the Department of Mathematics and Computer Science at Fuzhou University. His research interests include deep learning, knowledge representation and natural language processing.

Wenzhong Guo received the Ph.D. degree from the Department of Physics and Information Engineering, Fuzhou University, Fuzhou, China, in 2010. He was a Post-Doctoral Research Scholar with the Department of Computer Science, National University of Defense and Technology, Changsha, China, from 2011 to 2014, a Visiting Professor at the Faculty of Engineering, Information and System, University of Tsukuba, Japan, in 2013, and a Visiting Professor at the Department of Computer Science and Engineering, State University of New York at Buffalo, USA, in 2016. He is currently a Professor and the Director of the Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing. His research interests include the fields of data mining, machine learning, and artificial intelligence.