Asymmetric Self-Supervised Graph Neural Networks

Zhuo Tan, Bin Liu
Center of Statistical Research
Southwestern University of Finance and Economics
Chengdu, China
tanzhuo@smail.swufe.edu.cn, liubin@swufe.edu.cn

Guosheng Yin
Department of Statistics and Actuarial Science
University of Hong Kong
Hong Kong, China
gyin@hku.hk

***Abstract*—Despite self-supervised learning (SSL) having been successfully applied to graph data using graph neural networks (GNNs), most existing methods only consider undirected graphs where relationships among connected nodes are two-way symmetric (i.e., information can be passed back and forth). However, there is a vast amount of applications where the information flow is asymmetric, leading to directed graphs where information can only be passed in one direction. For example, a directed edge indicates that the information can only be conveyed forwardly from the start node to the end node, but not backwardly. To accommodate such an asymmetric structure of directed graphs, we propose a simple yet remarkably effective SSL framework for directed graph analysis to incorporate such one-way information passing. We define an incoming embedding and an outgoing embedding for each node to model its sending and receiving features respectively. Then a proposed auxiliary SSL task tries to predict the existence of the directed edges with the incoming and outgoing embedding of nodes. The auxiliary SSL task is jointly trained with a downstream primary task that updates nodes' incoming features and outgoing features accordingly with labels. Extensive experiments on multiple real-world directed graphs demonstrate outstanding performances of the proposed self-supervised GNNs in both node-level and graph-level tasks.**

***Index Terms*—graph neural networks, self-supervised learning, directed graph**

I. Introduction

Self-supervised learning (SSL) has been raised as a new paradigm to help with the demand for large amounts of labeled data [1]. SSL deliberately designs pretext tasks that use certain properties of input data itself as additional supervision to conduct supervised learning. The pretext tasks should be properly crafted to help the model learn task-related features [2]. In computer vision, various pretext tasks have been studied, including relative positions of picture patches [3] and augmented images (e.g., rotating, cropping, or resizing) prediction [4]. And in natural language processing, predicting the masked word in BERT [5] is a frequent pretext task.

The empirical successes of SSL have prompted researchers to extend it to the field of graph analysis, specifically in the area of graph neural networks (GNNs). A large number of recent works have attempted to combine GNNs and SSL [6]. The existing SSL-based GNNs can be classified into two categories based on the different graph pretext tasks, one is graph contrastive learning [7], the other is graph predictive learning [8]–[11]. Generally, graph contrastive learning aims to apply different operations onto an original graph to obtain its multiple views and then make an agreement between the representations of each view, such as maximizing the mutual information between each view [12], [13]. Graph predictive learning, in contrast, tries to reconstruct the adjacency matrix based on the output node embedding of GNNs [8] or predict the multiple randomly corrupted feature [14]. However, most of the existing SSL-based GNNs are designed for undirected graph analysis or simplifying a directed graph problem with undirected settings [15]. As we know, the directed graphs have a more complicated local structure than the undirected graphs. They arise broadly across different domains; for instance, email networks [16], citation networks [17], natural language sentences [18], as well as website networks [19], etc. These examples involve directional edges, indicating receiving and sending information in one-way or two-way flow among nodes. Hence, it is necessary to design a pretext task to exploit the asymmetric information passing on the directed graphs.

We believe that the asymmetry of the directed graph is a double-edged sword, complicating the problem on one hand while also providing us with valuable information for self-supervision on the other. That is, the directed graph-specific information that preserves the asymmetric structure [20]–[25] can be exploited to benefit SSL.

To this end, we propose to predict the directional edges with asymmetric embedding as an auxiliary task of SSL. Specifically, we introduce two types of embedding [20], an outgoing vector and an incoming vector (of the same dimension), for each node. The incoming embedding aims to capture receiving features and the outgoing embedding aims to capture sending-out features. Fig. 1 (b) illustrates an example of such asymmetric node representations, where node v in a directed graph is described by both an incoming vector $\mathbf{r}_v$ and an outgoing vector $\mathbf{s}_v$. And the existence of the edge from v to q should be determined by the outgoing embedding $\mathbf{s}_v$ of node v and incoming embedding $\mathbf{r}_q$ of node q. In particular, we model this process by $f(\mathbf{s}_v, \mathbf{r}_q)$, and $f(\cdot, \cdot)$ could be any appropriate link functions.

With the setting of two representations of each node, we develop an asymmetric message passing mechanism to update the incoming and outgoing embeddings and capture the asymmetric structure of directed graphs. Different from the traditional GNNs, the proposed asymmetric message passing has two aggregating/updating steps for node embedding updating, the first one to aggregate/update the receiving features of nodes and the second one to aggregate/update sending-out features. In practice, the auxiliary SSL task is jointly trained with a supervised downstream task with labels, such as a node-level or graph-level task.

The contributions of this work are as follows:

- We propose a novel directed self-supervised graph neural network, called asymmetric self-supervised GNNs (ASSGNN), with the incoming and outgoing embeddings to learn the different roles for each node in directed graphs;
- A novel self-supervised task for directed graphs is proposed to guide the learning of GNNs and alleviate the over-smoothing issue;
- Our approach works well on different directed graph datasets in semi-supervised nodes classification and graph regression tasks and achieves better performance than the state-of-the-art directed GNN methods.

The rest of the paper is organized as follows. Section II introduces the related work in self-supervised and directed GNNs. Section III formulates the basic GNN and the self-supervised graph reconstruction models for undirected graphs. The detailed formulation of our proposed ASSGNN is given in Section IV. The comparisons with the state-of-the-art methods and ablation study are shown in Section V. Section VI concludes the paper with remarks.

II. Related work

A. Self-supervised graph neural networks

Self-supervised Learning has shown great potential in learning representation from unlabeled data, that has been successfully applied to computer vision [3], [4] and natural language processing [5], [26]. Recently, lots of efforts have been done to extend SSL to graph data analysis under the framework of GNNs. There are two basic architectures of SSL-based GNNs, graph contrastive learning and graph predictive learning [6]. In general, contrastive learning tries to obtain multiple views of an input graph by applying different graph transformations on it, such as drop edge [27], drop node [28]. Then a well-designed contrastive object tries to agree with each view. The other framework of SSL-based GNNs is graph predictive learning [6], which is actually a supervised learning task where the informative labels are made out of the structure of the graph itself. For example, the label could be the existence of edge [15], [29] or the properties of graph [30], [31]. However, most of the existing methods of SSL-based GNNs focus on undirected graphs. The directed graph has an asymmetric structure, which is more complex than undirected graphs. We think the asymmetry of directed graphs brings us more opportunities to generate informative labels.

B. Directed graph neural networks

A majority of GNNs transform directed graphs to undirected ones by relaxing its direction structure, i.e., trivially adding edges to symmetrize the adjacency matrices. Monti, Otness, and Bronstein [32] deal with directed graphs by exploiting local graph motifs [33], which define a new set of symmetric motif adjacency matrices. Ma et al. [16] and Tong et al. [17] leverage Perron–Frobenius theorem and the stationary distribution of strongly connected graphs to generalize the Laplacians of undirected graphs to directed graphs. They also eliminate the directed structure to guarantee that the normalized Laplacian is symmetric. Tong et al. [17] further exploit the kth-order proximity between two nodes in a digraph by the inception network [34], which not only allows the model to learn features of different sizes within one convolutional layer but also obtains larger receptive fields.

Several works try to learn the specific structure of directed graphs. In [35] and [36], the incident matrix is used to combine the attributes of the incoming edges, outgoing edges, and vertex itself to update the vertex and combine the attributes of the source nodes, target nodes and edge itself to update the edge. Zhang et al. [19] encode the undirected geometric structure in the magnitude of its entries and directional information in the phase of its entries based on a complex Hermitian matrix known as the magnetic Laplacian. Thost and Chen [37] use the attention mechanism to only aggregate directed predecessors' information from the current layer and GRU to combine the neighbor and previous representation. Beani et al. [38] exploit vector flows over graphs to develop a globally consistent directional and asymmetric aggregation function. They define a vector field in the graph by use of the Laplacian eigenvectors and develop a method of applying directional derivatives and smoothing by projecting node-specific messages into the field.

III. Preliminaries

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with n nodes, where $\mathcal{V} = \{1, \ldots, n\}$ is the set of n nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges connecting paired nodes in $\mathcal{V}$. The graph structure is represented by the $n \times n$ symmetric adjacency matrix $\mathbf{A} = \{a_{ij}\}_{i,j=1}^{n}$, where $a_{ij} = 1$ if there exists an edge $e_{ij} \in \mathcal{E}$ between nodes i and j, and $a_{ij} = 0$ otherwise. The degree matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ is a diagonal matrix, where $\mathbf{D}_{ii} = \sum_i a_{ij} = \sum_j a_{ij}$. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote the feature matrix of the n nodes if graph $\mathcal{G}$ is associated with attributes. If the node attributes are not given, we can encode nodes as a one-hot vector, that is, $\mathbf{X} = \mathbf{I}$, where $\mathbf{I} \in \mathbb{R}^{n \times n}$ is an identity matrix.

A. Graph neural networks

GNNs learn node or edge representation vectors by using the graph structure $\mathbf{A}$ and node feature matrix $\mathbf{X}$. To facilitate subsequent discussions, we formalize an L-layers GNN under the architecture of message passing neural networks (MPNNs) [39], which computes the l-th layer representation vector $\mathbf{h}_v^l$ for node v in a graph $\mathcal{G}$ as

$$\begin{aligned} \mathbf{m}_v^l &= \text{AGGREGATE}^l \left(\left\{ \mathbf{h}_u^{l-1} \mid u \in \mathcal{N}(v) \right\} \right) \\ \mathbf{h}_v^l &= \text{COMBINE}^l \left(\mathbf{h}_v^{l-1}, \mathbf{m}_v^l \right), \quad l = 1, \ldots, L, \end{aligned}$$

where $\mathbf{h}_u^0 = \mathbf{X}_u \in \mathbb{R}^d$ is the input feature of node u, $\mathcal{N}(v)$ is a set of neighbors adjacent to v in the graph. Under the framework of MPNNs, messages flow from $\mathcal{N}(v)$ to v using an aggregation function, and then $\mathbf{h}_v^l$ of node v is updated based on the corresponding neighbor messages $\mathbf{m}_v^l$ from the previous message aggregation step.

Many existing methods [40]–[43] are specific examples of the MPNNs framework and the differences lie in the choices of AGGREGATE and COMBINE functions.

For the node-level task, the node representation $\mathbf{h}_v^L$ of the final layer L is used for prediction. For the graph-level task, the READOUT function computes a feature vector for the whole graph using some readout function according to

$$\mathbf{h}_{\mathcal{G}} = \text{READOUT}\left(\left\{h_v^L, v \in \mathcal{V}\right\}\right),$$

where READOUT can be a simple permutation invariant function such as summation or a more sophisticated graph-level pooling function [44], [45].

B. Self-supervised graph reconstruction

Graph reconstruction-based SSL aims to reconstruct specific properties of a given graph, has been used as a self-supervised auxiliary task for the training of GNNs in various studies [9], [10]. For an undirected graph, reconstructing the adjacency matrix (link prediction) could be taken as an auxiliary task of SSL as follows,

$$\begin{aligned} \mathbf{H} &= \text{GNNs}(\mathbf{A}, \mathbf{X}) \\ \hat{\mathbf{A}} &= g(\mathbf{H}), \end{aligned} \tag{1}$$

where $\mathbf{H}$ is the output feature of GNNs. $g(\cdot)$ is a reconstruction function (e.g., $g(\mathbf{H}) = \sigma(\mathbf{H}\mathbf{W}\mathbf{W}^T\mathbf{H}^T)$ [10]), where $\mathbf{W}$ is learnable parameters and σ is the sigmoid activation function.

Then a binary cross-entropy loss is used as the SSL optimization objective,

$$\mathcal{L}_{ssl} = -\sum_{ij} a_{ij} \log \hat{a}_{ij} + (1 - a_{ij}) \log(1 - \hat{a}_{ij}),$$

where $\hat{a}_{ij}$ is the ij-th element of the reconstruction adjacency matrix $\hat{\mathbf{A}}$.

IV. Method

In this section, we introduce the proposed model ASS-GNN (as shown in Fig. 1). Firstly, we design asymmetric graph reconstruction as a self-supervised auxiliary task for directed graphs. In addition, a directed graph neural network, asymmetric message passing, is proposed. Following that, two specialized downstream tasks, that is semi-supervised node classification and graph regression, are introduced. Finally, the primary downstream task is jointly trained with the self-supervised auxiliary task.

A. Asymmetric self-supervised graph reconstruction

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed graph with n nodes, we have an asymmetric adjacency matrix $\mathbf{A} = \{a_{ij}\}_{i,j=1}^n$, where $a_{ij} = 1$ if there exists a directed edge e_{ij} emanating from node i to node j, and $a_{ij} = 0$ otherwise. Specifically, each node can be split into an out-node and an in-node, and the rows of $\mathbf{A}$ index the out-nodes and the columns of $\mathbf{A}$ index the in-nodes. Thus, the directed graph has two diagonal degree matrices: an in-degree matrix $\mathbf{P} \in \mathbb{R}^{n\times n}$ with $\mathbf{P}_{jj} = \sum_i^n a_{ij}$ and an out-degree matrix $\mathbf{O} \in \mathbb{R}^{n\times n}$ with $\mathbf{O}_{ii} = \sum_j^n a_{ij}$. Note that the in-degree and out degree of each node are equal in undirected graphs, because of the symmetric $\mathbf{A}$. The nodes are characterized by the feature matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ which is the same as undirected graphs.

In undirected graphs, relationships are symmetric, and thus a node is represented by a single vector embedding $\mathbf{h}$. Then, the existence of the edge e_{ij} can be modeled as,

$$p_{ij} = f(\mathbf{h}_i, \mathbf{h}_j),$$

which is a classical way of reconstructing the adjacent matrix $\mathbf{A}$ for the undirected graph. $f : \mathbb{R}^c \times \mathbb{R}^c \to \mathbb{R}$ is a nonlinear link function, for an instance, such the function $g(\cdot)$ in (1).

Directed graphs, in contrast, have asymmetric node relationships. In particular, some nodes can have edges of both in-taking (receiving information) and out-flowing (sending information) functions, or the coexistence of both directed and undirected edges, which would make GNNs to learn the graph representation more difficult than an undirected setting. However, the asymmetric relationship can be a two-edged sword. It complicates our model but provides us with valuable information for self-supervised representation learning.

To capture the informative asymmetric characteristics, we propose to represent node v in a directed graph with an outgoing (sending information) embedding vector $\mathbf{s}_v$ and an incoming (receiving information) embedding vector $\mathbf{r}_v$, as shown in Fig. 1 (b).

With the asymmetric node embeddings for directed graphs (proposed in Section IV-B), the likelihood (existence) of a directional edge can be calculated based on the outgoing embedding of its source node and the incoming embedding of its target node. As shown by the example in Fig. 1 (b), the occurrence of the directional edge e_{vq}, from v to q, can be interpreted together by the vectors $\mathbf{s}_v$ and $\mathbf{r}_q$. In general, considering the outgoing embedding set $\{\mathbf{s}_i\}_{i=1}^n$ and the incoming embedding set $\{\mathbf{r}_j\}_{j=1}^n$ of the final layer (i.e., L-th layer) for all the nodes (for notation simplicity, we let $\mathbf{s}_i := \mathbf{s}_i^L$ and $\mathbf{r}_j := \mathbf{r}_j^L$), we formulate the existence of an edge $i \to j$ with the similarity between $\mathbf{s}_i$ and $\mathbf{r}_j$ followed by a sigmoid function as [10],

$$p_{ij} = P(a_{ij} = 1) = f(\mathbf{s}_i, \mathbf{r}_j) = \sigma\left(\mathbf{s}_i^T \mathbf{W}^T \mathbf{W} \mathbf{r}_j\right), \tag{2}$$

where $\mathbf{W} \in \mathbb{R}^{m\times c}$ is the learnable parameter, and $\mathbf{s}_i, \mathbf{r}_j \in \mathbb{R}^c$ are the final outgoing embedding of node i and the final incoming embedding of node j that output from a L-layer directed GNNs.

Hence, graph $\mathcal{G}$ can be reconstructed based on the joint likelihood of all edges. Consequently, the likelihood function of $\mathcal{G}$ is

$$\begin{aligned} P(\mathcal{G}) &= \prod_{i,j=1}^n p_{ij}^{a_{ij}} (1 - p_{ij})^{1-a_{ij}} \\ &= \prod_{i,j=1}^n \frac{\exp\left(\mathbf{s}_i^T \mathbf{W}^T \mathbf{W} \mathbf{r}_j a_{ij}\right)}{1 + \exp\left(\mathbf{s}_i^T \mathbf{W}^T \mathbf{W} \mathbf{r}_j\right)}. \end{aligned}$$

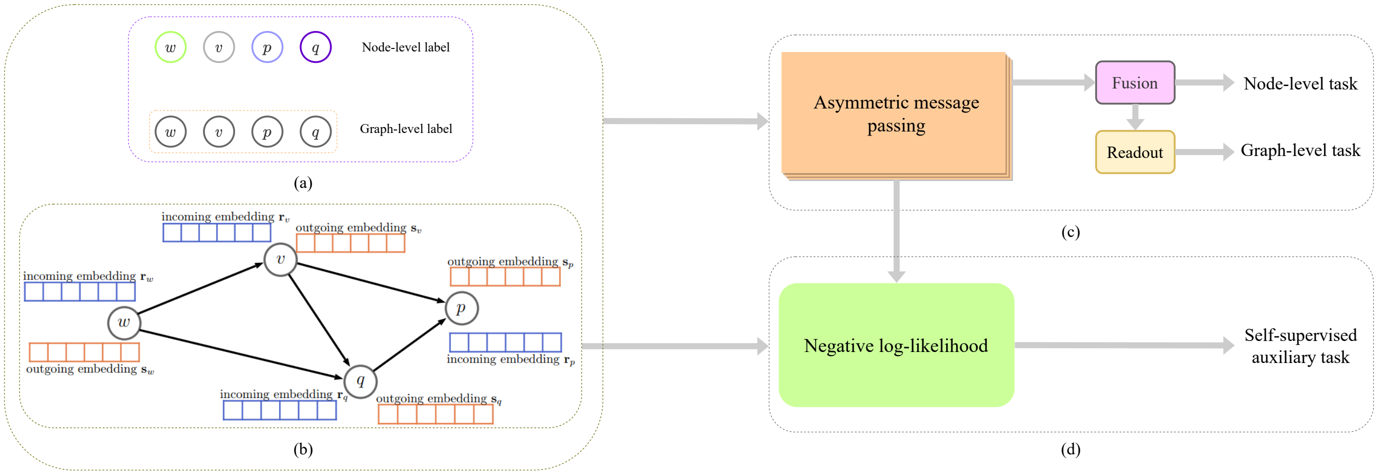
Node-level label
Graph-level label
(a)
incoming embedding $\mathbf{r}_v$
outgoing embedding $\mathbf{s}_v$
incoming embedding $\mathbf{r}_w$
outgoing embedding $\mathbf{s}_p$
outgoing embedding $\mathbf{s}_w$
incoming embedding $\mathbf{r}_p$
incoming embedding $\mathbf{r}_q$
outgoing embedding $\mathbf{s}_q$
(b)
Asymmetric message passing
Fusion
Node-level task
Readout
Graph-level task
(c)
Negative log-likelihood
Self-supervised auxiliary task
(d)

Fig. 1. Illustration of the model architecture. (a): Labels for the supervised tasks, with the node-level label corresponding to the node-level task and the graph-level label corresponding to the graph-level task; (b) Nodes with outgoing embedding and incoming embedding. For example, node v has both the outgoing embedding $\mathbf{s}_v$ and incoming embedding $\mathbf{r}_v$, which describe the sending and receiving pattern respectively. Node w has two outgoing edges but no incoming edge, hence its incoming embedding $\mathbf{r}_w = \mathbf{0}$. Likewise, the outgoing embedding $\mathbf{s}_p = \mathbf{0}$ because node p only contains incoming edges. (c) In the main supervised task, after asymmetric message passing (proposed in Section IV-B), the two embeddings are combined using a fusion function or an except readout function to be employed in node-level or graph-level downstream tasks. (d) The occurrence of a directional edge can be interpreted by the two types of embedding. For instance, the likelihood of edge $w \rightarrow v$ depends on $\mathbf{s}_w$ and $\mathbf{r}_v$. The negative log-likelihood of directed edges is formatted as the self-supervised auxiliary task. Downstream supervised tasks (panel (c), e.g., node-level or graph-level task) require initial embeddings (e.g. given node features), graph structure, and labels as inputs, whereas self-supervised auxiliary task (as shown in (d)) only needs the incoming and outgoing embeddings from asymmetric message passing, as well as graph structure, but not labels.

This leads to the self-supervised loss, a regularized negative log-likelihood of the existence of graph $\mathcal{G}$,

$$\begin{aligned}\mathcal{L}_{\text{ssl}}(\mathbf{s},\mathbf{r}) &= -\frac{1}{n^2}\log P(\mathcal{G}) \\ &= -\frac{1}{n^2}\sum_{i,j=1}^{n}\Big(\mathbf{s}_i^T\mathbf{W}^T\mathbf{W}\mathbf{r}_j a_{ij} \\ &\quad -\log\big(1+\exp\big(\mathbf{s}_i^T\mathbf{W}^T\mathbf{W}\mathbf{r}_j\big)\big)\Big).\end{aligned} \tag{3}$$

Otherwise, the embeddings of nodes would be similar due to larger receptive fields as the number of layers increases, which hinders the deeper GNNs model to take advantage of the long-term dependencies in the graph and is not conducive to the task. The proposed self-supervised auxiliary task, which equips the negative log-likelihood of directed edges in $\mathcal{G}$, maintains the similarity within one-hop neighbors, which can alleviate this over-smoothing problem.

B. Asymmetric message passing

In this section, we introduce the GNN-based feature extractor, which is an asymmetric message passing process (illustrated in Fig. 1 (c)). In the auxiliary SSL task, we propose to represent node v in a directed graph with an outgoing (sending information) embedding vector $\mathbf{s}_v$ and an incoming (receiving information) embedding vector $\mathbf{r}_v$, as shown in Fig. 1 (b). Based on the two asymmetric roles of each node in the directed graph, we develop a corresponding asymmetric message passing mechanism, which contains two steps with the first one to aggregate/update its incoming features and the second one to aggregate/update its outgoing features.

The main idea of asymmetric message passing is to process nodes according to the different behaviors in terms of sending and receiving patterns to learn receiving and sending feature maps, respectively.

We develop a multi-layer asymmetric message passing mechanism with the following rules,

$$\begin{aligned}\mathbf{m}_v^l &= \text{AGGREGATE}_1^l\left(\left\{\mathbf{r}_u^{l-1} \mid u \in \mathcal{N}_{in}(v)\right\}\right) \\ \mathbf{a}_v^l &= \text{AGGREGATE}_2^l\left(\left\{\mathbf{s}_w^{l-1} \mid w \in \mathcal{N}_{out}(v)\right\}\right) \\ \mathbf{r}_v^l &= \text{COMBINE}_1^l\left(\mathbf{r}_v^{l-1}, \mathbf{m}_v^l\right) \\ \mathbf{s}_v^l &= \text{COMBINE}_2^l\left(\mathbf{s}_v^{l-1}, \mathbf{a}_v^l\right),\end{aligned} \tag{4}$$

where $\mathbf{r}_v^0 = \mathbf{s}_v^0 = \mathbf{X}_v \in \mathbb{R}^d$ is the input feature of v, $\mathcal{N}_{in}(v)$ and $\mathcal{N}_{out}(v)$ are the sets of direct predecessors and successors of v respectively. More specifically, for $\forall u \in \mathcal{V}$, if there exists an edge $u \rightarrow v$, then $u \in \mathcal{N}_{in}(v)$; likewise, for $\forall w \in \mathcal{V}$, if there exists an edge $v \rightarrow w$, then $w \in \mathcal{N}_{out}(v)$. $l = 1, 2, \ldots, L$ denotes the layer of convolution. Finally, we can model the different roles of receiving and sending information, and obtain two embeddings of one node (i.e., the receiving messages flow from $\mathcal{N}_{in}(v)$ to the incoming embedding $\mathbf{r}_v$ and the sending messages flow from $\mathcal{N}_{out}(v)$ to the outgoing embedding $\mathbf{s}_v$ for node v, respectively).

For the node-level task, the two node representations $\mathbf{r}_v^L$ and $\mathbf{s}_v^L$ of the final layer are merged and used for prediction,

$$\mathbf{z}_v = \text{COMBINE}_3\left(\mathbf{r}_v^L, \mathbf{s}_v^L\right), \tag{5}$$

where L is the number of layers. For graph-level task, the readout phase computes a feature vector for the whole graph according to

$$\mathbf{z}_{\mathcal{G}} = \text{READOUT}\left(\{\mathbf{z}_v, v \in \mathcal{V}\}\right).$$

By defining the specific AGGREGATE and COMBINE functions, we would obtain a corresponding directed GNN method. For instance, consider the element-wise *mean* pooling in the framework of the asymmetric message passing, it is a directed version of GCN,

$$\begin{aligned}\mathbf{S}^l &= \sigma\left(\tilde{\mathbf{A}}\mathbf{S}^{l-1}\mathbf{W}_1^l\right)\\ \mathbf{R}^l &= \sigma\left(\hat{\mathbf{A}}\mathbf{R}^{l-1}\mathbf{W}_2^l\right),\end{aligned} \tag{6}$$

where $\mathbf{S}^0 = \mathbf{R}^0 = \mathbf{X} \in \mathbf{R}^{n\times d}$, σ is the activation function (i.e., ReLU), $\tilde{\mathbf{A}} = \mathbf{O}^{-1/2}(\mathbf{A}+\mathbf{I})\mathbf{P}^{-1/2} \in \mathbb{R}^{n\times n}$ is the normalized adjacency matrix of the directed graph $\mathcal{G}$ with added self-connections, $\hat{\mathbf{A}} = \mathbf{P}^{-1/2}(\mathbf{A}^T+\mathbf{I})\mathbf{O}^{-1/2} \in \mathbb{R}^{n\times n}$ and $\mathbf{W}_1^l, \mathbf{W}_2^l \in \mathbb{R}^{d_{l-1}\times d_l}$ are trainable parameters. $\mathbf{S}^l$ and $\mathbf{R}^l$ are the sending and receiving embedding of the l-th layer in matrix form. Through (6), we can aggregate (i.e., *mean*) the predecessor (or successor) and root nodes to matrix $\mathbf{R}^l$ (or $\mathbf{S}^l$). Consistent with (2), we let $\mathbf{S} := \mathbf{S}^L$ and $\mathbf{R} := \mathbf{R}^L$ for notation simplicity, where $\mathbf{S}^L \in \mathbb{R}^{n\times c}$ and $\mathbf{R}^L \in \mathbb{R}^{n\times c}$ are the outgoing and incoming embeddings of the final L-th layer.

For node classification task, we can fusion the incoming and outgoing embedding vectors, followed by a softmax function to map every node to a probability distribution,

$$\mathbf{Z} = \text{Softmax}\left(\mathbf{R}+\mathbf{S}\right) \in \mathbb{R}^{n\times c},$$

where c denotes the number of classes.

For graph regression, we can use a *sum* pooling to obtain the graph-level representation task,

$$\mathbf{Z}_{\mathcal{G}} = \text{SUM}\left(\mathbf{Z}\right) \in \mathbb{R}^c.$$

C. Downstream tasks

We aim to learn a powerful feature extractor that projects each node to a c-dimensional space $\mathbb{R}^c$ which is beneficial for the specific task. In this section, we introduce how to learn the task-specific feature extractor, i.e., the optimization objective of supervised downstream tasks.

Formally, the obtained node representations are used to minimize the supervised error over all labeled examples. For the semi-supervised multi-class classification task, we use the cross-entropy error,

$$\mathcal{L}_{\text{sup}} = -\sum_{l\in\mathcal{Y}_L}\sum_{c=1}^{C}\mathbf{Y}_{lc}\ln\mathbf{Z}_{lc},$$

where $\mathcal{Y}_L$ is the set of node indices that have labels, $\mathbf{Y}_{lc}$ is the true label, $\mathbf{Z}_{lc}$ is the predicted label and C is the number of classes. For the graph regression task, we use the mean square error,

$$\mathcal{L}_{\text{sup}} = \frac{1}{|\mathbf{Y}_{\mathcal{G}}|}\sum_i \|\mathbf{Z}_{i,\mathcal{G}} - \mathbf{Y}_{i,\mathcal{G}}\|^2,$$

where $\mathbf{Y}_{\mathcal{G}}$ is the true graph label of training set and $\mathbf{Z}_{\mathcal{G}}$ is the predicted graph label.

With the $\mathbf{S}$ and $\mathbf{R}$ are given in (6), the self-supervised learning loss (i.e., Equation (3)) can be reformulated as follow,

$$\begin{aligned}\mathcal{L}_{\text{ssl}}(\mathbf{S},\mathbf{R}) = -\frac{1}{n^2}\mathbf{1}^T\Big(&\mathbf{S}\mathbf{W}\mathbf{W}^T\mathbf{R}^T \odot \mathbf{A}\\ &- \log\left(1+\exp(\mathbf{S}\mathbf{W}\mathbf{W}^T\mathbf{R}^T)\right)\Big)\mathbf{1},\end{aligned}$$

where $\mathbf{W} \in \mathbb{R}^{c\times m}$ is the learnable parameter, $\mathbf{1}$ denotes the vector with all entries being 1, and $\odot$ denotes the Hadamard product.

Finally, we combine the supervised task and the self-supervised task to jointly train and obtain the total loss function,

$$\mathcal{L} = \mathcal{L}_{\text{sup}} + \lambda\mathcal{L}_{\text{ssl}}, \tag{7}$$

where λ is a hyper-parameter as the trade-off weights for combining the supervised loss $\mathcal{L}_{\text{sup}}$ and the self-supervised loss $\mathcal{L}_{\text{ssl}}$.

Actually, the above self-supervised auxiliary loss $\mathcal{L}_{\text{ssl}}$ can be interpreted as a regularization term of the primary loss $\mathcal{L}_{\text{sup}}$. This regularization forces nodes to maintain the virginity that they are just similar to their direct neighborhoods and avoid the over-smoothing problem in GNNs [46].

Furthermore, the proposed method is very flexible and can be extended to undirected graphs by setting $\mathbf{r}_v = \mathbf{s}_v$ for each node v.

V. Experiment

We evaluate the effectiveness of our model on both node-level and graph-level supervised downstream tasks. Node-level experiments are performed on citation and co-purchase datasets, graph-level on a neural architecture dataset. Semi-supervised node classification is a common task of GNNs, which can be used to verify the learning capability of our model. Another task is graph regression, which can be used to verify that our model is not node-level-specific and can be easily adapted to the graph-level task. Compared with the canonical experiments for undirected graphs, the main difference lies in that the task under our setup is applied to the directed graphs and the given adjacency matrix $\mathbf{A}$ is asymmetric. All experiments are conducted on PyTorch [1] and Pytorch-Geometric [47], under a computer environment with one Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz CPU, four NVIDIA Titans V GPU cards and Ubuntu 16.04 System.

A. Experimental settings

TABLE I
DETAILED INFORMATION ON THE FOUR REAL DATASETS. WE CHOOSE 20 LABELS PER CLASS FOR THE TRAINING SET FOLLOWING DIGCN [17].

Datasets	Nodes	Edges	Classes	Features	Label rate
Cora-ML	2995	8416	7	2879	4.67%
Citeseer	3312	4715	6	3703	3.62%
Am-Computer	13752	287209	10	767	1.45%
Am-Photo	7650	143663	8	745	2.10%

[1]https://pytorch.org

TABLE II
OVERALL ACCURACY COMPARISON ON NODE CLASSIFICATION BETWEEN THE PROPOSED ASSGNN, ASSGNN W/O AUX (ASSGNN WITHOUT THE SSL AUXILIARY TASK), AND SEVEN EXISTING METHODS. THE BEST RESULTS ARE HIGHLIGHTED IN **BOLDFACE** AND THE SECOND IN *Italian font*. / DENOTES OUT OF MEMORY.

Models	Cora-ML	Citeseer	Am-Computer	Am-Photo
GCN	53.11 ± 0.8	54.36 ± 0.5	60.50 ± 1.6	53.20 ± 0.4
SGC	51.14 ± 0.6	44.07 ± 3.5	76.17 ± 0.1	71.25 ± 1.3
APPNP	70.07 ± 1.1	65.39 ± 0.9	63.16 ± 1.4	79.37 ± 0.9
GraphSage	72.06 ± 0.9	63.19 ± 0.7	79.29 ± 1.3	87.57 ± 0.9
GAT	71.91 ± 0.9	63.03 ± 0.6	79.45 ± 1.5	89.10 ± 0.7
DGCN	75.02 ± 0.5	66.00 ± 0.4	/	83.66 ± 0.8
DiGCN	80.28 ± 0.5	66.11 ± 0.7	**85.94 ± 0.5**	90.02 ± 0.5
ASSGNN	**80.76 ± 0.2**	**69.21 ± 0.3**	*84.34 ± 0.7*	**90.70 ± 0.3**
ASSGNN w/o Aux	*80.52 ± 0.3*	*69.07 ± 0.3*	84.27 ± 0.7	*90.67 ± 0.3*

1) Datasets: For the semi-supervised node classification task, we apply the ASSGNN on four directed graph datasets with detailed information given in Table I. The four real datasets include two citation datasets: Cora-ML [48] and Citeseer [49], as well as two Amazon co-purchase datasets [50]: AM-Computer and AM-Photo. In our experiments, we randomly split the datasets and perform multiple experiments for obtaining stable and reliable results. The mean and standard deviation of classification accuracy are calculated by running each experiment 20 times with random weight initialization. For the training/validation/testing split, we follow the rules in DiGCN [17] to choose 20 labels per class for the training set, 500 labels for the validation set, and the rest for the testing set.

For the graph regression task, we apply our model on NA dataset [51]. The NA dataset contains 19020 neural architectures from the ENAS software. The task is to evaluate each neural architecture's weight-sharing accuracy [52] (a proxy of the true accuracy) on CIFAR-10 [2]. Since it is a regression task, we use the RMSE, MAE, and MAPE metrics. Following [37], we randomly split the dataset into 90% training and 10% held-out test sets.

2) Baselines: We compare our model with seven state-of-the-art models that can be divided into three main categories: (i) spectral-based GNNs including GCN [40], SGC [53], APPNP [54]; (ii) spatial-based GNNs including GraphSage [15] and GAT [42]; and (iii) directed GNNs including DGCN [55] and DiGCN [17]. The detailed descriptions of baseline methods are summarized in the Supplementary Material.

3) Implementation details: For the node classification task, the settings of all baseline models followed by DiGCN [17]. For the graph regression task, all baselines and ASSGNN are set to the same value and followed by DAGNN [37]. The activation function is ReLU. Detailed hyper-parameter settings are shown in Supplementary Material.

For our experiments, including node-level and graph-level tasks, the dropout rate is 0.5, and the activation function is ReLU. For citation datasets, Cora-ML and Citeseer, we train our model for a maximum of 1000 epochs with early stopping if the validation accuracy does not increase for 200 consecutive epochs, and the hidden layer dimension is 64. For Amazon co-purchase datasets, AM-Computer and AM-Photo, we train our model for a maximum of 500 epochs without early stopping, and the hidden layer dimension is 128. Note that we normalize features in Cora-ML and Citeseer datasets followed by DiGCN [17]. For the NA dataset, we train our model for a maximum of 100 epochs, and the learning rate is 0.001. The activation function is ReLU. The hidden layer dimension is 64, and the batch size is 32. All baselines and ASSGNN are set to the same value and followed by DAGNN [37]. Detailed hyper-parameter settings are shown in Supplementary Material.

[2]An image classification dataset. http://www.cs.toronto.edu/~kriz/cifar.html

TABLE III
ACCURACY COMPARISON ON GRAPH REGRESSION TASK BETWEEN OUR ASSGNN AND SIX EXISTING METHODS. THE BEST RESULTS ARE HIGHLIGHTED IN **BOLDFACE** AND THE SECOND IN *Italian font*.

Models	RMSE	MAE	MAPE
GCN	*0.0051*	*0.0040*	*0.0055*
SGC	0.0100	0.0081	0.0110
APPNP	0.0060	0.0046	0.0063
GraphSage	0.0053	0.0041	0.0056
GAT	0.0055	0.0042	0.0057
DiGCN	0.0299	0.0293	0.0397
ASSGNN	**0.0050**	**0.0039**	**0.0053**

B. Experimental results

1) Comparisons with the state-of-the-art methods on node classification: It can be seen from Table II that our method achieves state-of-the-art classification accuracy on citation datasets and comparable results on Amazon co-purchase datasets. The spectral-based models, including GCN and SGC, perform poorly on directed graph datasets in contrast to their better performances in undirected graphs. The possible explanation is that these models are limited to aggregating features from the direct successors using asymmetric adjacency matrices so that they can only capture the sending information but not the receiving information. APPNP is an exception because it allows features to be propagated randomly with a certain teleport probability, which breaks the path restriction and achieves good performance in directed graphs. Both GraphSage and GAT are spatial-based methods, which yield better performances than APPNP, demonstrating that these methods are more suitable for directed graphs. GraphSage generates embeddings by sampling and aggregating features from nodes' local neighborhoods, which can capture more information than direct successors. GAT specifies different weights for different neighborhoods, which reflects differences in the edges. DGCN performs well on three datasets, while it leads to out-of-memory on AM-Computer because the method uses both the first- and second-order proximity matrices to obtain structural features in directed graphs. DiGCN performs better than our model only on the Amazon-Computer dataset, which may be due to the use of the inception method. However, as shown by the results on the other three datasets, our model performs better than all the existing methods. And despite removing the self-supervised auxiliary task (proposed in Section IV-A), our model still outperforms the state-of-the-art methods, demonstrating that the incoming embedding and outgoing embedding can effectively accommodate the asymmetric structure of directed graphs. To put it another way, the self-supervised auxiliary task improves the accuracy of the model on the semi-supervised node classification task.

2) Comparisons with the state-of-the-art methods on graph regression: Table III shows that our model outperforms the six baseline models on the graph regression task. The results of Table III show that a simple structure will achieve better performance for the NA dataset. DiGCN performs poorly, most likely due to its complexity, which relies on the inception network to exploit kth-order proximity. Correspondingly, GCN performs well because of its lightweight structure. Furthermore, the activation function is necessary because GCN outperforms SGC. While GCN, GraphSage, and GAT work well, our model proves the superiority of capturing directed structure. Note that we use the same readout function (*sum* operation) in these experiments.

3) Accuracy with different hyper-parameter: We validate our model with different values of the hyper-parameter λ in (7). As shown in Fig. 2, despite the accuracy varies depending on the λ, the models with SSL auxiliary task (i.e., $\lambda \neq 0$) perform better than without ones (i.e., $\lambda = 0$) in the Cora-ML and Citeseer datasets, which proves the superiority of the proposed self-supervised strategy. The value of the hyper-parameter should be adjusted for different datasets because different graphs exhibit different neighborhood structures [54]. We choose λ according to the mean test accuracy of 20 times experiments.

4) Accuracy with different fusion functions: Fig. 3 illustrates the results with different fusion operations (i.e., the COMBINE$_3$ function mentioned in (5)) in the Citeseer and AM-Photo datasets, including Sum, Max, Mean, and Concatenate. Sum and Mean functions achieve better results in the proposed model because they require fewer parameters which can prevent the model from over-fitting and thus capture more information [43]. And they perform better in all layers, especially on the AM-Computer dataset. Concatenate performs worse because it requires additional linear parameters.

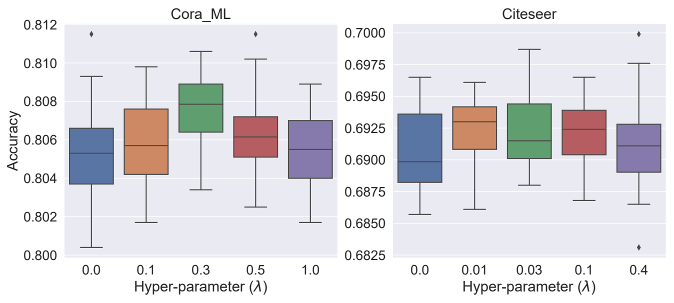

Fig. 2. Influence of the hyper-parameter λ on classification performance.

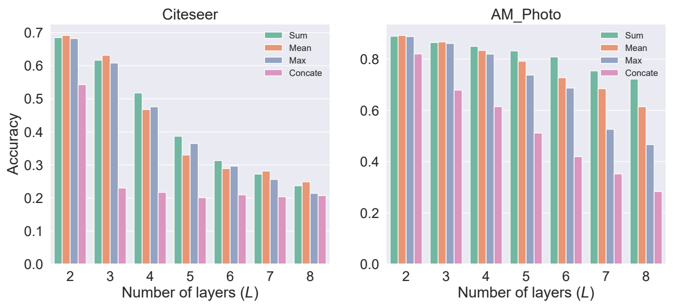

Fig. 3. Influence of fusion functions on classification performance. Different colors denote various fusion functions.

VI. Conclusion

We propose a simple yet effective SSL method to model the asymmetric property of the directed graph within the GNNs framework. The idea is to let each node play two asymmetric roles simultaneously: one is to act as an outgoing node, and another is to act as an incoming node. The outgoing and incoming roles are quantified by two different embedding vectors. The outgoing embedding aims to capture the sending features of a node, and the incoming embedding aims to model the receiving features. With the two embeddings of each node, any directional edge in the graph can be interpreted by the outgoing embedding of its source node and the incoming embedding of its target node. After that, a self-supervised auxiliary task is employed to predict the directional edges via asymmetric embeddings. Such one node but two roles setting can be easily applied to the current GNNs. We define two aggregating/updating steps for asymmetric message passing: one for aggregating/updating the receiving features of nodes, and another for aggregating/updating the sending features of nodes. Experimental results on multiple real-world datasets illustrate that the proposed asymmetric self-supervised GNNs yields good performance in directed graph downstream primary tasks (including node-level and graph-level tasks). In the future, we intend to put forward the theoretical properties of the proposed model from the perspective of graph signal processing.

VII. Acknowledgements

This work was partially supported by the Education and Teaching Reform Special Funds for the Central Universities of China (No.2021YJG025).

References

[1] Y. Xie, Z. Xu, Z. Wang, and S. Ji, "Self-supervised learning of graph neural networks: A unified review," *arXiv preprint arXiv:1905.13728*, 2021.

[2] K. Sun, Z. Lin, and Z. Zhu, "Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes," in *AAAI*, 2020, pp. 5892–5899.

[3] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *ECCV*, 2016, pp. 69–84.

[4] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Big Data*, vol. 6, p. 60, 2019.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019, pp. 4171–4186.

[6] Y. Xie, Z. Xu, J. Zhang, Z. Wang, and S. Ji, "Self-supervised learning of graph neural networks: A unified review," *arXiv preprint arXiv:2102.10757*, 2021.

[7] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations," *NeurIPS*, pp. 5812–5823, 2020.

[8] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[9] Q. Zhu, B. Du, and P. Yan, "Self-supervised training of graph convolutional networks," *arXiv preprint arXiv:2006.02380*, 2020.

[10] D. Kim and A. H. Oh, "How to find your friendly neighborhood: Graph attention design with self-supervision," in *ICLR*, 2021.

[11] W. Jin, T. Derr, Y. Wang, Y. Ma, Z. Liu, and J. Tang, "Node similarity preserving graph convolutional networks," in *WSDM*, 2021, pp. 148–156.

[12] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *ICLR*, 2018.

[13] F.-Y. Sun, J. Hoffman, V. Verma, and J. Tang, "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization," in *ICLR*, 2020.

[14] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "MGAE: marginalized graph autoencoder for graph clustering," in *CIKM*, 2017, pp. 889–898.

[15] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1025–1035.

[16] Y. Ma, J. Hao, Y. Yang, H. Li, J. Jin, and G. Chen, "Spectral-based graph convolutional network for directed graphs," *arXiv preprint arXiv:1907.08990*, 2019.

[17] Z. Tong, Y. Liang, C. Sun, X. Li, D. S. Rosenblum, and A. Lim, "Digraph inception convolutional networks," in *NeurIPS*, 2020, pp. 17 907–17 918.

[18] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," in *AAAI*, 2019, pp. 7370–7377.

[19] X. Zhang, N. Brugnone, M. Perlmutter, and M. J. Hirn, "Magnet: A magnetic neural network for directed graphs," *arXiv preprint arXiv:2102.11391*, 2021.

[20] K. Rohe, T. Qin, and B. Yu, "Co-clustering directed graphs to discover asymmetries and directional communities," *Proceedings of the National Academy of Sciences*, vol. 113, no. 45, pp. 12 679–12 684, 2016.

[21] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *SIGKDD*, 2016, pp. 1105–1114.

[22] C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao, "Scalable graph embedding for asymmetric proximity," in *AAAI*, 2017, pp. 2942–2948.

[23] M. Khosla, J. Leonhardt, W. Nejdl, and A. Anand, "Node representation learning for directed graphs," in *ECML PKDD*, 2019, pp. 395–411.

[24] J. Sun, B. Bandyopadhyay, A. Bashizade, J. Liang, P. Sadayappan, and S. Parthasarathy, "ATP: directed graph embedding with asymmetric transitivity preservation," in *AAAI*, 2019, pp. 265–272.

[25] S. Zhu, J. Li, H. Peng, S. Wang, and L. He, "Adversarial directed graph embedding," in *AAAI*, 2021, pp. 4741–4748.

[26] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," in *ICML*, 2014, pp. 1188–1196.

[27] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR*, 2020.

[28] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," in *NeurIPS*, 2020, pp. 22 092–22 103.

[29] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artifical Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[30] Y. Rong, Y. Bian, T. Xu, W. Xie, Y. Wei, W. Huang, and J. Huang, "Self-supervised graph transformer on large-scale molecular data," in *NeurIPS*, 2020, pp. 12 559–12 571.

[31] D. Hwang, J. Park, S. Kwon, K. Kim, J. Ha, and H. J. Kim, "Self-supervised auxiliary learning with meta-paths for heterogeneous graphs," in *NeurIPS*, 2020, pp. 10 294–10 305.

[32] F. Monti, K. Otness, and M. M. Bronstein, "MOTIFNET: A motif-based graph convolutional network for directed graphs," in *DSW*, 2018, pp. 225–228.

[33] A. R. Benson, D. F. Gleich, and J. Leskovec, "Higher-order organization of complex networks," *Science*, vol. 353, no. 6295, pp. 163–166, 2016.

[34] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016, pp. 2818–2826.

[35] L. Shi, Y. Zhang, J. Cheng, and H. Lu, "Skeleton-based action recognition with directed graph neural networks," in *CVPR*, 2019, pp. 7912–7921.

[36] B. Fu, S. Fu, L. Wang, Y. Dong, and Y. Ren, "Deep residual split directed graph convolutional neural networks for action recognition," *IEEE MultiMedia*, vol. 27, no. 4, pp. 9–17, 2020.

[37] V. Thost and J. Chen, "Directed acyclic graph neural networks," in *ICLR*, 2021.

[38] D. Beaini, S. Passaro, V. Létourneau, W. L. Hamilton, G. Corso, and P. Lió, "Directional graph networks," in *ICML*, 2021, pp. 748–758.

[39] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *ICML*, 2017, pp. 1263–1272.

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

[41] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," in *ICLR*, 2016.

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

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

[44] Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *NeurIPS*, 2018, pp. 4805–4815.

[45] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *AAAI*, 2018, pp. 4438–4445.

[46] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *ICML*, 2018, pp. 5449–5458.

[47] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[48] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking," in *ICLR*, 2018, pp. 1–13.

[49] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[50] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[51] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-VAE: A variational autoencoder for directed acyclic graphs," in *NeurIPS*, 2019, pp. 1586–1598.

[52] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proceedings of the Thirty-fifth International Conference on Machine Learning, ICML*, 2018, pp. 4095–4104.

[53] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *ICML*, 2019, pp. 6861–6871.

[54] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *ICLR*, 2019.

[55] Z. Tong, Y. Liang, C. Sun, D. S. Rosenblum, and A. Lim, "Directed graph convolutional network," *arXiv preprint arXiv:2004.13970*, 2020.