EAI Endorsed Transactions
on e-Learning

Research Article **EAI.EU**

A survey on graph neural networks

Jing Wang[1]*

[1]School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo, Henan 454000,P R China

Abstract

In recent years, we have witnessed the developments that deep learning has brought to machine learning. It has solved many problems in the areas of computer vision, speech recognition, natural language processing, and various other tasks with state-of-the-art performance. However, the data in these tasks is typically represented in Euclidean space. As technology develops, more and more applications are generating data from non-Euclidean domains and representing them as graphs with complex relationships and interdependencies between objects. This poses a significant challenge to deep learning algorithms. This is because, due to the uniqueness of graphs, applying deep learning to the ubiquitous graph data is not an easy task. To solve the problem in non-Euclidean domains, Graph Neural Networks (GNNs) have emerged. A Graph Neural Network (GNN) is a neural model that captures dependencies between graphs by passing messages between graph nodes. This paper introduces commonly used graph neural networks, their learning methods, and common datasets for graph neural networks. It also provides an outlook on the future of Graph Neural Networks.

Keywords: deep learning, graph neural networks, Euclidean domains, non-Euclidean domains, graph data

Received on 18 June 2023, accepted on 25 June 2023, published on 10 July 2023

doi: 10.4108/eetel.3466

1 Introduction

With the rapid development of neural networks in recent years, deep learning has become the "jewel" of artificial intelligence and machine learning[1]. Many machine learning tasks that once relied on manual methods to extract feature information (e.g., image recognition, machine translation) have been replaced by various more advanced deep learning methods. Of course, the success of deep learning in areas such as image classification, video processing, speech recognition, and natural language understanding is no accident, thanks not only to big data and high-performance computing power but also to the effectiveness of deep learning [2] itself in extracting potential representations from Euclidean data. For graphs can be regular or irregular. A graph may have both unordered nodes of different sizes, nodes from the same graph may have different numbers of neighbours, and each node in the graph may have different neighbourhoods. This leads to the fact that some operations of deep learning algorithms (such as convolution operations) work well in the Euclidean domain, but are difficult to apply to the graph domain.

Graphs are ubiquitous and widely used in the real world, and they can represent objects and their relationships in various fields. Examples include large-scale social networks, transportation networks, chemical molecular analysis, recommendation systems[3], ecosystems, and so on. More and more applications rely on representing data generated in non-Euclidean[4]domains as graphs with complex relationships and interdependencies between objects. The complexity of graph structures poses a major challenge to existing deep-learning algorithms. In recent years, people have a strong interest in deep learning methods for extended graph data, and the following graph neural network (GNN) has emerged, driven by deep learning algorithms such as convolutional neural network (CNN) and recurrent neural network (RNN). The emergence of graph neural networks makes it possible to apply deep learning algorithms to non-Euclidean domains to solve graph problems.

Graph Neural Networks (GNN) is a deep learning algorithm based on graph structure that learns the representation of nodes and edges in a graph, implementing

*Corresponding author. Email: wangjing@home.hpu.edu.cn tasks such as classification, clustering, and prediction of the graph as a whole. Unlike traditional machine learning algorithms that require the transformation of graphs into vectors or matrices, GNNs improve the representation of graph data by performing calculations directly on the graph, using the relationships between nodes. For example, in social network analysis, GNNs can help us discover community structure and predict user interests and behavior, among other tasks. In chemical molecular analysis, GNNs can help us with tasks such as classifying, clustering, and predicting molecules. In recommender systems, GNNs can use the relationships between users to improve the effectiveness of recommendations.

2 Background

Early research on graph neural networks (GNNs) belongs to the category of recurrent neural networks (RecGNNs) and has a high overhead. Sperduti and Starita [5] introduced neural networks to direct acyclic graphs and promoted the research of GNNs. Gori, Monfardini [6] first introduced the concept of graph neural networks. [7, 8]further elaborate the concept of graph neural networks.
In recent years, with the wide application of non-Euclidean data, more and more people focus on the study of graph neural networks. Wu, Pan [9] classifies graph neural networks into four categories. Zhang, Cui [10] A comprehensive review of deep learning methods on different types of graphs. Thomas, Moallemy-Oureh [11] Classifies graph neural networks according to their different abilities to process graph types and attributes. Waikhom and Patgiri [12] The learning mode of graph neural network is summarized. Zhou, Cui [13] A generic pipeline design for graph neural network models is proposed. There are also many research works on graph neural network learning methods. Cao, Li [14] extracted feature information in hyperspectral classification to avoid the problem of over-smoothing of message delivery caused by [15]. As the research work progressed, contrast-based learning methods were also successful. Okuda, Satoh [16] proposed unsupervised graph representation learning to discover common objects and a set of specific objects in an image for localisation. The node classification and edge detection of [17] combines two learning methods, random walk, and language modelling, and the learned representations can be used for downstream tasks.

This paper provides a comprehensive review of different models of graph neural networks and how graph neural networks learn. A more complete overview of graph neural networks is provided. In summary, the main contributions of this paper include: (1) a comprehensive and detailed review of models of graph neural networks; (2) a discussion of graph-based training approaches; and (3) challenges for future research on graph neural networks.

The rest of the paper includes: Section 3 introduces the concept and notation of graphs. Section 4 introduces the respective learning methods of graph convolutional networks (GCN), graph attention networks (GAT), and graph autoencoders (GAE). In addition, the difference between graph attention networks (GAT) and graph convolutional networks (GCN) is described. At the same time, we also make a simple distinction between GAT and GAN. Section 5 describes the datasets commonly used in graph neural networks. Section 6 summarises the paper and discusses the challenges faced by graph neural networks.

3 Graph Neural Network

3.1 Concept and notation representation of graphs

A graph neural network[18, 19] is a deep neural network suitable for the analysis of graph structures. The notation involved in this paper is interpreted as shown in Table 1. The graph is expressed as G=(V, E). where $v = \{v_1, v_2, v_3, .., v_n\}$ represents the set of N=|V| nodes. E ⊆ V × V represents the set of edges between nodes [20]. We use A $R^{N\times N}$ to represent the adjacency matrix [21]. an element of the ith row of A can be written as A(i, :) and an element of the jth column can be written as A(:, j). A(i, j) represents an element of the ith row and jth column of A.

3.2 Structure of Graph Neural Network

The graph structure is such that each node is defined by its own features and by the features of the nodes connected to it. The purpose of GNN[22] is to learn a state embedding vector $h_v \in R^s$ for each node[23], which contains information about each node's neighbours. h_v represents the state vector of the node . This vector can be used to generate output o_v .

(i) Assume that f (.) is a function with parameters called the local transition function, which is shared among all nodes and updates the node state based on input from neighbouring nodes.
(ii) Assume that g (.) is the local output function, which is used to describe how the output is generated.

$$h_v = f(x_v, x_{co[v]}, h_{ne[v]}, x_{ne[v]}) \tag{1}$$

$$o_v = g(h_v, x_v) \tag{2}$$

where, x_v represent the feature vector of node v, $x_{co[v]}$ represent the feature vector of the edge associated with node v, $h_{ne[v]}$ represent the state vector of the neighbouring nodes of node v . $x_{ne[v]}$ represent the feature vector of the neighbouring nodes of node v.
If all the state vectors, output vectors, feature vectors and vectors obtained from node features are superimposed and

represent by H , O , X , X_N respectively, then a more compact representation can be obtained as follows:

Table 1 GNN common notation

Type	Explanation
G = (V, E)	A graph
N, M	The number of nodes and edges
$V = \{v_1, \ldots, v_N\}$	The set of nodes
F^V , F^E	The attributes/features of nodes and edges
A	The adjacency matrix
$\mathbf{D}(i,i) = \sum_j \mathbf{A}(i,j)$	The diagonal degree matrix
L = D − A	The Laplacian matrix
$\mathbf{Q}\Lambda\mathbf{Q}^T = \mathbf{L}$	The eigendecomposition of L
$P = D^{-1}A$	The transition matrix
$\mathrm{N}_k(i), \mathrm{N}(i)$	The k-step and 1-step neighbors of v_i
$\dot{\mathbf{H}}^{l}$	The hidden representation in the lth layer
f_l	The dimensionality of $\dot{\mathbf{H}}^{l}$
ρ(·)	Some non-linear activation function
$\mathbf{X}_1$ □ $\mathbf{X}_2$	The element-wise multiplication
Θ	Learnable parameters
s	The sample size

$$H = F(H, X) \tag{3}$$

$$O = G(H, X_N) \tag{4}$$

Where, F and G are respectively called the global transfer function and the global output function and are stacked versions of f and g for all nodes in the graph. According to Banach's immobility point theorem, GNN uses the following conventional iterative approach to calculate the state covariates:

$$H^{t+1} = F(H^t, X) \tag{5}$$

where , H^t represent the tensor of the tth iteration cycle of H. For arbitrary initial values H_0 ,Eq. (5) can be obtained by fast convergence to the final fixed-point solution of Eq. (3).

4 Graph Neural Network Models

4.1 Graph Convolutional Network (GCN□

GCN is a convolutional neural network that acts directly on the graph and makes use of its structural information. The main idea of GCN[24-26] is that for each node, we consider all of its neighbors and the characteristic information it contains. Assuming that we use the average () function, this is done for each node to obtain an average representation that can be fed into the neural network. Modern GCNs mimic CNNs by designing convolution and readout functions to learn common local and global structural patterns of graphs. We first discuss the convolution operation and then move to the readout operation and some other improvements. Convolutional neural networks[27] play a central role in building many other complex GNN models.

Graph Convolution Method

Graph convolutional neural networks include spatially based graph convolutional neural networks[28-30] and spectral based graph convolutional neural networks[31-34].

(1) Spatially based graph convolutional neural networks

The spatial domain-based graph convolutional neural structure consists of three main types of operators: neighbour sampling[35], message computation and message aggregation. The graph convolutional neural structure based on spatial domain mainly consists of three types of operators: neighbor sampling, message computation and information aggregation. In GCN, an aggregate operation is used to aggregate adjacent nodes represented by a node to achieve message transmission between nodes. Figure 1 shows the transfer of node information based on spatial domain GCN.

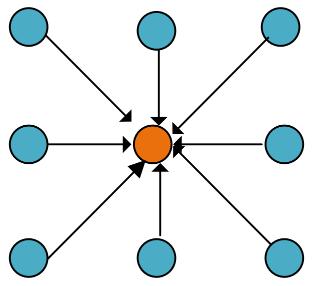

Figure 1: GCN node information transmission based on spatial domain

The simplest aggregation process is to do a product operation of the node features of the graph (X) with the topological structure information of the graph (adjacency matrix A). The exact process is shown in Figure 2.

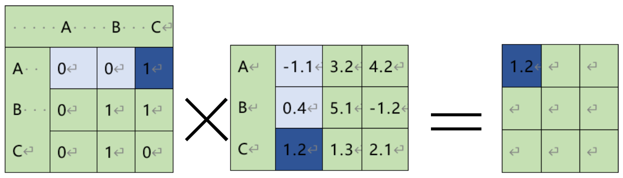

Figure 2: Information aggregation process for spatial domain-based GCNs

To solve the problem in Figure 2 of not calculating the nodes' own features and aggregating them directly by summation, which can cause the gradients to explode or disappear, we can add the unit matrix I to the adjacency matrix A and aggregate the features of the neighbouring nodes by taking a weighted average.

According to the different methods of convolutional stacking, space-based GCN can be further divided into two categories: recurrent-based and composition-based spatial GCN. recurrent-based approaches use the same graph convolution layer to update the hidden representation, and compositional -based approaches use a different graph convolution layer to update the hidden representation. Figure 3 illustrates this difference.

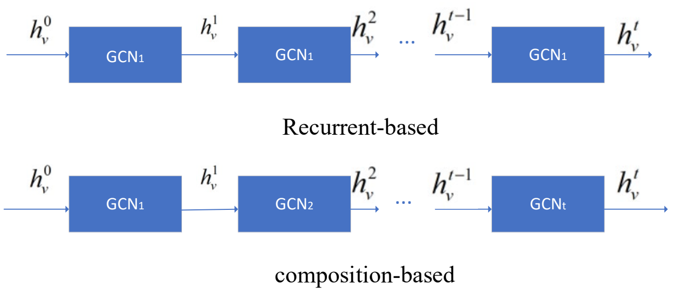

Figure 3: Comparison of recurrent-based and composition-based

The spatial approach is to define the convolution directly in the spatial domain. The problem faced is that, because each node's neighbours are of different sizes, it is impossible to define a neighbourhood of the same size, so achieving parameter sharing faces greater difficulties, but the idea is still that the convolution is still a weighted average of a node over its neighbouring nodes, so many subsequent approaches aim to solve the problem of parameter sharing.

(2)Spectral-based graph convolutional neural network

Convolution based on spectral methods is a special case of convolution based on spatial methods. Spectral domain-based graph convolution via neural networks investigates the properties of graphs with the help of the eigenvalues and eigenvectors of the Laplacian matrix of the graph. Filters are introduced to define convolution from a signal processing perspective. Firstly, the signal in the spectral domain is multiplied using the theorem of convolution. Secondly, the Fourier transform is used to transform the signal to the original space to achieve convolution. This approach avoids the difficulty of defining convolution caused by the fact that the graph data does not satisfy translation invariance. Because the structure of the graph does not satisfy translation invariance, it is not possible to define convolution directly in the spatial domain, so the signal is transformed into the frequency domain, where the convolution operation is realised, before being transformed back into the spatial domain, which is the spectral method. Graph convolutional neural networks based on spectral methods assume that the graph is undirected. The normalised graph Laplacian matrix is a mathematical representation of an undirected graph, defined as:

$$\mathbf{L} = \mathbf{I}_\mathbf{n} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \tag{6}$$

where D represents the diagonal matrix of node degrees, A represents the adjacency matrix of the graph. Using the symmetric positive semidefinite property of the graph Laplacian matrix, the normalised Laplacian matrix can be decomposed as:

$$\mathrm{L} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T \tag{7}$$

Where, $\mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_{n-1}] \in \mathbf{R}^{n\times n}$ is Feature vector matrix, Λ is the diagonal matrix of eigenvalues (spectrum), $\boldsymbol{\Lambda}_{ii} = \lambda_i$. The feature vectors of the regularised Laplacian matrix form a set of orthogonal bases. In graph signal processing, the signal of a graph $x \in \dot{R}^N$ is a feature vector consisting of the individual nodes of the graph, x_i representing the ith node. The Fourier transform and Fourier inverse transform of a graph X are defined as:

$$F(x) = U^T x \tag{8}$$

$$F^{-1}(\hat{x}) = \mathrm{U}\hat{\mathrm{x}} \tag{9}$$

Where, $\overline{x}$ is the result of the Fourier transform. To better understand the Fourier transform of a graph, we can see from its definition that it does project the input graph signal into an orthogonal space whose base is made up of the eigenvectors of the regularised graph Laplacian. The elements of the transformed signal are the coordinates of the graph signal in the new space, so that the original input signal can be expressed as:

$$\mathrm{x} = \sum_i \hat{\mathrm{x}}_i \mathrm{u}_i \tag{10}$$

This is the result of the Fourier inverse transform. Next we can define the graph convolution operation on the input signal X.

$$x *_G g = F^{-1}(\mathcal{F}(x) \odot \mathcal{F}(g)) = \mathrm{U}(\mathrm{U}^T \mathrm{x} \odot \mathrm{U}^T \mathrm{g}) \quad (11)$$

Where, $g \in R^N$ is the filter we define; $\odot$ Indicates the Hadamard product. Suppose we define such a filter[36]:

$$\mathbf{g}_\theta = diag(\mathbf{U}^T \mathbf{g}) \quad (12)$$

Thus, the graph convolution operation[37] can be represented in a simplified way as:

$$x *_G g_\theta = \mathbf{U} \mathbf{g}_\theta \mathbf{U}^T \mathbf{x} \quad (13)$$

Spectral-based graph convolution networks all follow this pattern, with the key difference between them being the choice of filter. The following models of spectral-based graph convolution networks exist: Spectral CNN, Chebyshev[31] Spectral CNN (ChebNet), Adaptive Graph Convolution Network (AGCN).

Comparison Between Spectral and Spatial Models

As the earliest graph convolutional networks, spectral-based models have achieved impressive results in many graph-related analysis tasks. These models have some theoretical basis in graph signal processing. By designing new graph signal filters, we can theoretically design new graph convolutional networks. However, spectral-based models have some insurmountable drawbacks, which we will address below in terms of efficiency, generality and flexibility.

In terms of efficiency, the computational cost of spectral-based models increases dramatically with the size of the graph, as they either need to perform feature vector calculations or process the entire graph at the same time, making them difficult to apply to large graphs. Space-based models have the potential to handle large graphs as they perform convolution directly in the graph domain by aggregating neighbouring nodes. The computation can be performed in a batch of nodes rather than in the whole graph. Sampling techniques can be introduced to improve efficiency when the number of neighbouring nodes increases.

In terms of generality, spectral-based models assume a fixed graph, making it difficult for them to add new nodes to the graph. On the other hand, space-based models perform graph convolution locally at each node and can easily share weights between different locations and structures.

In terms of flexibility, the spectral-based model is limited to working on undirected graphs; the Laplace matrix on directed graphs is not clearly defined, so the only way to apply the spectral-based model to directed graphs is to convert the directed graph to an undirected graph. Space-based models are more flexible in dealing with multiple source inputs which can be combined into aggregation functions. As a result, spatial models have received increasing attention in recent years.

4.2 Graph Attention Network(GAT）

Graph Attention Network (GAT) [38] consists of a number of functionally identical blocks (Graph Attention Layer) [39]. Its properties include high efficiency, low storage type, inductive learning and full graph access. The graph attention layer has a feature value of $\vec{\mathbf{h}} = \{\vec{h}_1, \vec{h}_2, \cdots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$ for the node at input. where N represents the number of nodes and F represents the dimensionality of the node features. After a Graph Attention Layer, a new feature vector is output, which can be represented as $\vec{\mathbf{h}'} = \{\vec{h'}_1, \vec{h'}_2, \ldots, \vec{h'}_N\}, \vec{h'}_i \in \mathbb{R}^{F'}$, assuming that the dimension of the node feature of this feature vector is F'. As shown in Figure 4.

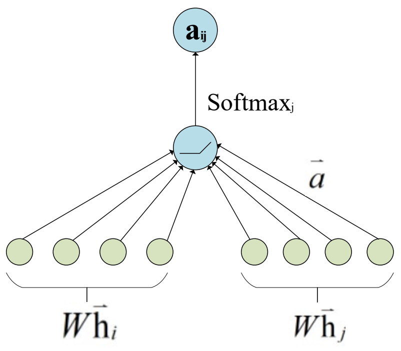

Figure 4: Attention layer in GAT

The purpose of using Self-attention is to improve the expressiveness of $\vec{\mathbf{h}'}$. In the Graph Attention Layer, a weight matrix $\mathbf{W} \in \mathbb{R}^{F' \times F}$ is first applied to each node using a weight matrix, and then self-attention is used for each node to calculate an attention coefficient, the shared self-attention mechanism used here, denoted a:

$$e_{ij} = a(\mathbf{W}\vec{\mathbf{h}}_\mathbf{i}, \mathbf{W}\vec{h}_j) \quad (14)$$

e_{ij} represents the importance of node j for node i. In theory we can calculate the weight of any node in the graph to the central node. In GAT, to simplify the calculation, the nodes are restricted to the one-hop neighbours of the central node, and in addition the nodes take themselves into account as neighbouring nodes. In the existing studies a there are many ways to choose. For example, choosing a single-layer feedforward network with parameter $\vec{a} \in \mathbb{R}^{2F'}$ and then using LeakyReLU to do a non-linearisation gives.

$$e_{ij} = \text{LeakerReLU}(\vec{\mathbf{a}}^{T}\left[\mathbf{W}\vec{h}_i \square \mathbf{Wh}_j\right]) \quad (15)$$

Finally, the neighbouring nodes of the central node are normalized using softmax:

$$\alpha_{ij} = \text{softmax}j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathrm{N}_i} \exp(e_{ik})} \quad (16)$$

The output feature $\vec{\mathbf{h}'}$ is obtained by weighting the input features.

$$\vec{h}_i' = \sigma\left(\sum_{j \in \mathrm{N}_i} \alpha_{ij} \vec{h}_j\right) \quad (17)$$

In order to improve the generalisation of the attention mechanism, GAT chose to use a multi-headed attention layer, i.e. a single-headed attention[40] layer from a set of K mutually independent graph attention layers, and then stitch their results together. At this point, $\mathbf{h}_i'$ is:

$$\mathbf{h}_i' = \square_{k=1}^{K} \sigma\left(\sum v_j \in \tilde{\mathrm{N}}(v_i) \alpha_{ij}^{(k)} \mathbf{W}^k \mathbf{h}_j\right) \quad (18)$$

where ‖ represents the splicing operation, $\alpha_{ij}^{(k)}$ represents the weight factor calculated from the kth group of attention mechanisms, and $\mathbf{W}^k$ is the weight factor of the kth module. In order to reduce the dimensionality of the feature vector, we can also use the averaging operation instead of the splicing operation, as shown in the following equation.

$$\vec{h}_i' = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in \mathrm{N}j} \alpha_{ij}^{k} \mathbf{W}^k \vec{h}_j\right) \quad (19)$$

4.3 Graph Autoencoder(GAE)

Starting with the graph-based self-encoder proposed in Kipf and Welling [41], graph self-encoders have come in handy in many fields due to their simple encoder-decoder[42, 43] structure and efficient encode capability. Figure 5 briefly describes the flow of a graph self-encoder (GAE).

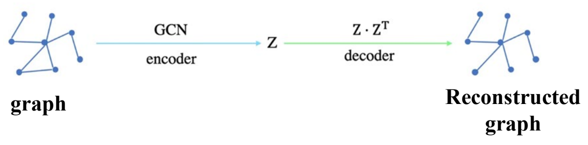

Figure 5: GAE workflow

Encoder

GAE[44, 45] uses the GCN as an encoder[46] to obtain latent representations (or embedding) of the nodes, a process that can be expressed in a short line of equation [47]:

$$\mathbf{Z} = \mathbf{GCN}(\mathbf{X}, \mathbf{A}) \quad (20)$$

Using the GCN as a function, X and A are input to the GCN as a function and the output $\mathbf{Z} \in \mathrm{R}^{N \times f}$, Z represents the latent representations (embedding) of all nodes. The GCN is defined as follows:

$$\text{GCN}(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\text{ReLU}(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0)\mathbf{W}_1 \quad (21)$$

where $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, W_0 and W_1 represent the parameters to be learned. Here the GCN is equivalent to a function with the node features [48, 49] and adjacency matrix as input and the node embedding[50] as output, with the aim of obtaining the embedding only.

Decoder

GAE uses the inner-product as decoder to reconstruct the original graph:

$$\hat{\mathbf{A}} = \sigma(\mathbf{Z}\mathbf{Z}^{\mathrm{T}}) \quad (22)$$

$\hat{\mathbf{A}}$ represents the reconstructed adjacency matrix. A good Z should make the reconstructed adjacency matrix[51] as similar as possible to the original adjacency matrix, because the adjacency matrix determines the structure of the graph. Therefore, GAE uses cross-entropy as the loss function[52] in the training process.

$$\mathrm{L} = -\frac{1}{N}\sum y \log \hat{y} + (1-y)\log(1-\hat{y}) \quad (23)$$

where y represents the value of an element in the adjacency matrix A (0 or 1) and $\hat{y}$ represents the value of the corresponding element in the reconstructed adjacency matrix $\hat{A}$ (between 0 and 1). It can be seen from the loss function that the closer and more similar the reconstructed adjacency matrix (or reconstructed graph) is to the original adjacency matrix (or original graph), the better.

4.4 Differences and connections between GCN and GAT

The core difference between GAT and GCN is how to collect and accumulate feature representations of neighbouring nodes at a distance of 1. GAT replaces the fixed normalization operations in GCN with an attention mechanism. Essentially, GAT simply replaces the standard function of GCN with a feature aggregation function of neighbouring nodes using attention weights.GAT was created to address the shortcomings of GCN.

Disadvantages of GCN include:

i. The weights assigned to different neighbours on the same order neighbourhood are identical, and it is not possible to assign different weights to different nodes in the neighbourhood. This limits the model's ability to capture the relevance of spatial information and is the reason why it is inferior to GAT for many tasks.

ii.The way in which the GCN combines features of adjacent nodes is dependent on the structure of the graph, which makes the trained model relatively poor at generalising to graphs of other structures.

Benefits of the GAT include:

i.Different weights can be assigned to different nodes in the neighbourhood.

ii.After the attention mechanism is introduced, it is only relevant to neighboring nodes (nodes with shared edges), and there is no need to get information about the whole graph:(1) The graph need not be undirected (if the edge does not exist, we can simply omit the calculation;(2) It makes our technique directly applicable to Inductive Learning - including the task of evaluating models graphically that are completely invisible during training.

The classification process of GAT is very similar to that of GCN in that it uses a softmax function + cross-entropy loss function + gradient descent to complete the process. In essence: GCN and GAT both aggregate the features of neighbouring vertices to the central vertex (an aggregate operation) and use the local stationary on the graph to learn the new vertex feature representation. The difference is that GCN uses the Laplacian matrix and GAT uses the attention coefficients. To a certain extent, GAT is stronger because the correlation between vertex features is better integrated into the model.

In contrast to GCN, GAT is suitable for inductive tasks, where the important learning parameters are W and a(.), which only relate to vertex features and have nothing to do with the structure of the graph. Therefore, changing the structure of the graph in a test task has little effect on GAT, and only requires changing Ni and recalculating it. In contrast, GCN is a graph-wide calculation, where the node features of the whole graph are updated in a single calculation. The parameters learned are largely related to the graph structure, which puts GCN in a difficult position for the inductive task.

In addition it is important to note that although the GAT and GAN[53-56] appears to be only one letter difference, but the actual meaning to indeed. GAT is a kind of graph neural network, but GAN is not. The following is a brief introduction to GAN to give us a clearer idea of the difference between GAT and GAN.

GAN [53-56]is composed of two neural networks. The first one is called the Discriminator, D(Y). It takes input Y(such as a graph) and outputs a value that indicates whether Y looks "real." D(Y) can be thought of as some kind of energy function that is close to 0 when Y is a real sample, and positive when Y is noisy or strange [57-59]. The other network is called the Generator (G (Z)). Here Z is usually a vector randomly sampled from a simple distribution (e.g., Gaussian distribution[60]), and the generator G(Z) is used to generate pictures, which are then used to train the discriminator D(Y)(to give lower values to real pictures and higher values to other pictures). In the process of training D, give it a real picture, make it adjust the parameter output lower value; Give it a picture of G and ask it to adjust the parameters to output a larger value D(G(Z)). On the other hand, as G is trained, it adjusts its internal parameters to make the images it produces more and more realistic. That is, it has been optimizing the images it produces to fool D into thinking that the images it produces are real.

This means that for these generated images, G wants to minimise the output of D, while D wants to maximise the output of D. The two networks have opposite aims and are in an adversarial posture. This is called adversarial training, or GAN.

In the following we will explain the training process of G AN in relation to the formula. First, a generator neural net work is built, with all parameters represented by θ. The pu rpose is to generate images x, and these samples x all obey a distribution $P_G(x;\theta)$. Then, n images are drawn from an existing database of images, corresponding to n points in a high-dimensional space $\{x_1, x_2, \ldots, x_n\}$. The action of "drawing" is equivalent to sampling in distribution $P_{data}(x)$, and the probability of being able to draw $\{x_1, x_2, \ldots, x_n\}$, i.e. $P_{data}(x_1), P_{data}(x_2), \ldots, P_{data}(x_n)$, is large. The goal of the generator training is to get $P_G(x)$ and $P_{data}(x)$ as close as possible. We therefore want $P_G(x_1;\theta), P_G(x_2;\theta), \ldots, P_G(x_n;\theta)$ each of these probabilities to be large, in other words, we want $\prod_{k=1}^{n} P_G(x_k;\theta)$ to be as large as possible. Finally, we train the network on the parameters found.

$$\theta^* = \arg\max_{\theta} \prod_{k=1}^{n} P_G(x_k;\theta) \tag{24}$$

where $\prod_{k=1}^{n} P_G(x_k;\theta)$ is the Likelihood of the sample. due to

$$\arg\max_{\theta} \prod_{k=1}^{n} P_G(x_k;\theta) = \arg\min_{\theta}[KL(P_{data} \mid\mid P_G)] \tag{25}$$

where KL refers to KL Divergence, which can indicate the closeness of two distributions. The equation above says that maximising Likelihood and minimising KL Divergence mean the same thing. So this step becomes: train the network to find the parameters.

$$\theta^* = \arg\min_{\theta}[KL(P_{data} \mid\mid P_G)] \tag{26}$$

5 Datasets

In recent years, commonly used datasets for graph neural networks include Cora, Citeseer, PubMed. Table 2 provides a comparison of these three datasets

Table 2 Comparison of datasets

Category	Cora	Citeseer	PubMed
Nodes	2708	3327	19717
Edges	5429	4732	44338
Features	1433	3703	500
Classes	7	6	3

Cora dataset
The Cora [61]dataset consists of machine learning papers, It divides the thesis into seven categories: Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning and Theory. Papers are selected in such a way that each paper is cited or cited by at least one other paper in the final corpus. There are 2708 papers in the corpus. After stem extraction and removal of stop words, we are left with only 1433 words of unique size. Delete all words with document frequency less than 10.

The dataset contains two files, cora.conten and cora.cite. The content of cora.conten is described in the following format: <paper_id> <word_attributes>+ <class_label>. The first entry in each line (paper_id) is a unique numbered ID for each paper, the subsequent (word_attributes) contains 1433 binary codes indicating whether each word in the vocabulary is present (represented by a 1) or absent (represented by a 0) in the paper, and the last entry (class_label) indicates the class label of the paper. And cora.cite contains the citation relations of the papers in the corpus in the format <ID of cited paper> <ID of citing paper>. Each row of data contains the coded IDs of two papers, the first entry (ID of cited paper) indicates the number of the cited paper and the second entry (ID of citing paper) indicates the number of the citing paper.

(2)Citeseer dataset
The Citeseer dataset is a selection of papers from the CiteSeer library of digital papers, classified into six categories: Agents, AI, DB, IR, ML, HCI. Papers were selected in such a way that each paper in the final corpus cited or was cited by at least one other paper. There are 3327 papers in the entire corpus. After stem extraction and removal of stop words, only 3703 words remained. All words with a document frequency of less than 10 were removed.

The dataset contains two files, citeseer.conten and citeseer.cites. The contents of citeseer.conten are in the format <paper_id> <word_attributes> + <class_label>. The first entry in each line (paper_id) is a unique numbered ID for each paper, the subsequent (word_attributes) contains 3703 binary codes indicating whether each word in the vocabulary is present (represented by a 1) or absent (represented by a 0) in the paper, and the last entry (class_label) indicates the class label of the paper. And citeseer.cites contains the citation relationships of the papers in the corpus in the format <ID of cited paper> <ID of citing paper>. Each row of data contains the coded IDs of two papers, with the first entry (ID of cited paper) indicating the number of the cited paper and the second entry (ID of citing paper) indicating the number of the citing paper.

(3)PubMed dataset
The PubMed dataset consists of 19717 scientific publications on diabetes from the Pubmed database, divided into three categories: Diabetes Mellitus, Experimental, Diabetes Mellitus Type 1, and Diabetes Mellitus Type 2. The citation network consists of 44,338 links. Each publication in the dataset is described by a TF/IDF weighted word vector in a dictionary of 500 unique words.TF-IDF (term frequency-inverse document frequency) is a common weighting technique used in information retrieval and data mining.TF is the word frequency (TF-IDF is a statistical method for assessing the importance of a word to a collection of documents or to one of the documents in a corpus. The importance of a word increases proportionally with the number of times it appears in a document, but decreases inversely with the frequency with which it appears in the corpus.

The dataset consists of three files: □ Pubmed-Diabetes.NODE.paper.tab. Its content format is described as <paper_id> +<label=****> +< TF-IDF>. The first entry of each line of data (paper_id) is the unique numbering ID of each paper, the second entry is "label=***","***" indicates the category of the paper, followed by 500 floating point TF_IDF values, in the form of "word=***"," word" indicates the term, "***" indicates the TF_IDF value of the term. (2) t Pubmed-Diabetes.GRAPH.pubmed.tab. This file is useless and you do not need to pay attention to it. (3) Pubmed-Diabetes.DIRECTED.cites.tab.

6 Conclusion

Firstly, this paper introduces four commonly used graph neural networks and their respective learning methods. Secondly, this paper also introduces the datasets that have been commonly used for graph neural networks in recent years.

Graph neural networks are very promising and have a wide range of applications in areas such as social network analysis, recommender systems, biomedicine and visualisation. At present, as research continues to progress, graph neural network technology is also evolving and new models and algorithms are emerging. Therefore, we can foresee that graph neural networks will be more widely and deeply used in future research and applications. However, current graph neural networks are currently facing many challenges. For example, (1) the challenge of processing scaled graph data. As the scale of graph data continues to increase, how to efficiently process graph data storage, sampling, acquisition and transmission has become an important challenge for graph neural networks. (2) The challenge of model interpretability. As with other deep learning models, the black-box nature of graph neural networks makes model interpretability an issue that cannot be ignored. How to better explain the decision process and results of the model to help users better understand the model and thus better use it is an important challenge for graph neural networks. (3) The challenge of data sparsity. Unlike other deep learning models, the data that graph neural networks need to handle is usually sparse, which poses a great challenge to the training and inference of the models. How to better handle sparse data and improve the performance and efficiency of the model is another important challenge facing graph neural networks. In the future, research on graph neural networks will have to be considered in conjunction with these issues.

References

1. LeCun, Y., Y. Bengio, and G. Hinton, *Deep learning.* Nature, 2015. **521**(7553): p. 436-44.
2. Zhang, Y.-D., Z. Dong, and C. Cattani, *Deep learning for computer-aided medical diagnosis.* Multimedia Tools and Applications, 2020. **79**(21): p. 15073-15073.
3. Wu, S., et al., *Session-Based Recommendation with Graph Neural Networks.* 2019.
4. Michael, et al., *Geometric Deep Learning: Going beyond Euclidean data.* IEEE Signal Processing Magazine, 2017. **34**(4): p. 18-42.
5. Sperduti, A. and A. Starita, *Supervised neural networks for the classification of structures.* IEEE Transactions on Neural Networks, 1997. **8**(3): p. 714.
6. Gori, M., G. Monfardini, and F. Scarselli. *A new model for learning in graph domains*. in *IEEE International Joint Conference on Neural Networks*. 2005.
7. Scarselli, F., et al., *The Graph Neural Network Model.* IEEE Transactions on Neural Networks, 2009. **20**(1): p. 61.
8. Gallicchio, C. and A. Micheli. *Graph Echo State Networks*. in *International Joint Conference on Neural Networks*. 2010.
9. Wu, Z., et al., *A Comprehensive Survey on Graph Neural Networks.* IEEE transactions on neural networks and learning systems, 2021(1): p. 32.
10. Zhang, Z., P. Cui, and W. Zhu, *Deep Learning on Graphs: A Survey.* 2018.
11. Thomas, J.M., et al., *Graph Neural Networks Designed for Different Graph Types: A Survey.* 2022.
12. Waikhom, L. and R. Patgiri, *Graph Neural Networks: Methods, Applications, and Opportunities.* 2021.
13. Zhou, J., et al., *Graph neural networks: A review of methods and applications.* AI Open, 2020. **1**: p. 57-81.
14. Cao, Z., X. Li, and L. Zhao, *Unsupervised Feature Learning by Autoencoder and Prototypical Contrastive Learning for Hyperspectral Classification.* 2020.
15. Yang, L., et al. *Toward Unsupervised Graph Neural Network: Interactive Clustering and Embedding via Optimal Transport*. in *2020 IEEE International Conference on Data Mining (ICDM)*. 2020.
16. Okuda, M., et al. *Unsupervised Common Particular Object Discovery and Localization by Analyzing a Match Graph*. in *IEEE International Conference on Acoustics, Speech and Signal Processing*. 2021.
17. Du, L., et al. *Dynamic Network Embedding : An Extended Approach for Skip-gram based Network Embedding*. in *Twenty-Seventh International Joint Conference on Artificial Intelligence {IJCAI-18}*. 2018.
18. Liu, R., et al., *Federated Graph Neural Networks: Overview, Techniques and Challenges.* 2022.
19. Zheng, X., et al., *Graph Neural Networks for Graphs with Heterophily: A Survey.* 2022.
20. Wang, S.-H., *Covid-19 classification by FGCNet with deep feature fusion from graph convolutional network and convolutional neural network.* Information Fusion, 2021. **67**: p. 208-229.
21. Zhou, L., et al. *A Weighted GCN with Logical Adjacency Matrix for Relation Extraction*. in *ECAI 2020 - 24th European Conference on Artificial Intelligence*. 2020.
22. You, J., et al., *Identity-aware Graph Neural Networks.* 2021.
23. Guttery, D.S., *Improved Breast Cancer Classification Through Combining Graph Convolutional Network and Convolutional Neural Network.* Information Processing and Management, 2021. **58**.
24. Atwood, J. and D. Towsley, *Diffusion-Convolutional Neural Networks.* Computer Science, 2015.
25. Niepert, M., M. Ahmed, and K. Kutzkov, *Learning Convolutional Neural Networks for Graphs.* JMLR.org, 2016.
26. Gilmer, J., et al., *Neural Message Passing for Quantum Chemistry.* 2017.
27. Kipf, T.N. and M. Welling, *Semi-Supervised Classification with Graph Convolutional Networks.* 2016.
28. Monti, F., et al. *Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs*. in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.
29. Liu, Z., et al., *GeniePath: Graph Neural Networks with Adaptive Receptive Paths.* Proceedings of the AAAI Conference on Artificial Intelligence, 2018. **33**.
30. Gao, H., Z. Wang, and S. Ji, *Large-Scale Learnable Graph Convolutional Networks*. 2018, ACM.
31. Defferrard, M., X. Bresson, and P. Vandergheynst, *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*. 2016.
32. Henaff, M., J. Bruna, and Y. Lecun, *Deep Convolutional Networks on Graph-Structured Data.* Computer Science, 2015.
33. Li, R., et al., *Adaptive Graph Convolutional Neural Networks*. 2018.
34. Bianchi, F.M., et al., *Graph Neural Networks with Convolutional ARMA Filters.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021. **PP**(99): p. 1-1.
35. Dong, J., et al., *Global Neighbor Sampling for Mixed CPU-GPU Training on Giant Graphs.* 2021.
36. Wang, X., et al., *Neural Graph Collaborative Filtering.* ACM, 2019.
37. Krizhevsky, A., I. Sutskever, and G. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks.* Advances in neural information processing systems, 2012. **25**(2).
38. Velikovi, P., et al., *Graph Attention Networks.* 2017.
39. Vaswani, A., et al., *Attention Is All You Need.* arXiv, 2017.

40. Merity, S., *Single Headed Attention RNN: Stop Thinking With Your Head*. 2019.
41. Kipf, T.N. and M. Welling, *Variational Graph Auto-Encoders*. 2016.
42. Badrinarayanan, V., A. Kendall, and R. Cipolla, *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2017: p. 1-1.
43. Cho, K., et al., *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. Computer Science, 2014.
44. Liu, Z., D. Zhou, and J. He. *Towards Explainable Representation of Time-Evolving Graphs via Spatial-Temporal Graph Attention Networks*. in *Conference on Information and Knowledge Management*. 2019.
45. Schulman, J., et al., *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. Computer ence, 2015.
46. Ng, I., et al., *A Graph Autoencoder Approach to Causal Structure Learning*. 2019.
47. Satapathy, S.C., *Secondary pulmonary tuberculosis identification via pseudo-Zernike moment and deep stacked sparse autoencoder*. Journal of Grid Computing, 2022. **20**: p. 1.
48. Zhang, Y.-D., *Pseudo Zernike Moment and Deep Stacked Sparse Autoencoder for COVID-19 Diagnosis*. CMC-Computers, Materials & Continua, 2021. **69**(3): p. 3145–3162.
49. Wang, S.-H., *DSSAE: Deep Stacked Sparse Autoencoder Analytical Model for COVID-19 Diagnosis by Fractional Fourier Entropy*. ACM Transactions on Management Information Systems, 2021. **13**(1).
50. Peng, C., et al., *A Survey on Network Embedding*. IEEE Transactions on Knowledge and Data Engineering, 2017. **PP**(99): p. 1-1.
51. B, J.X.A., et al., *Attention adjacency matrix based graph convolutional networks for skeleton-based action recognition - ScienceDirect*. Neurocomputing, 2021. **440**: p. 230-239.
52. Dickson, M.C., A.S. Bosman, and K.M. Malan, *Hybridised Loss Functions for Improved Neural Network Generalisation*. 2022.
53. Mirza, M. and S. Osindero, *Conditional Generative Adversarial Nets*. Computer Science, 2014: p. 2672-2680.
54. Radford, A., L. Metz, and S. Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. Computer ence, 2015.
55. Salimans, T., et al., *Improved Techniques for Training GANs*. 2016.
56. Karras, T., S. Laine, and T. Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.
57. Zhang, Y., *Deep learning in food category recognition*. Information Fusion, 2023. **98**: p. 101859.
58. Wang, S., *Advances in data preprocessing for biomedical data fusion: an overview of the methods, challenges, and prospects*. Information Fusion, 2021. **76**: p. 376-421.
59. Zhang, Y.-D. and Z.-C. Dong, *Advances in multimodal data fusion in neuroimaging: Overview, challenges, and novel orientation*. Information Fusion, 2020. **64**: p. 149-187.
60. Mackay, D.J.C., *The Humble Gaussian Distribution*. 2006.
61. Mohan, R., *Collective Classification in Network Data*.