International Journal of Theoretical and Applied Research (IJTAR)
ISSN: 2812-5878

Homepage: https://ijtar.journals.ekb.eg

Original article

An Overview and Evaluation on Graph Neural Networks for Node Classification

Asmaa M. Mahmoud[1*], Abeer S. Desuky[1], Heba F. Eid[1], Hoda A. Ali[1]

[1] *Department of Mathematics, Faculty of Science, Al-Azhar University, Cairo, Egypt.*

ARTICLE INFO

Received 09/08/2023
Revised 21/01/2024
Accepted 31/01/2024

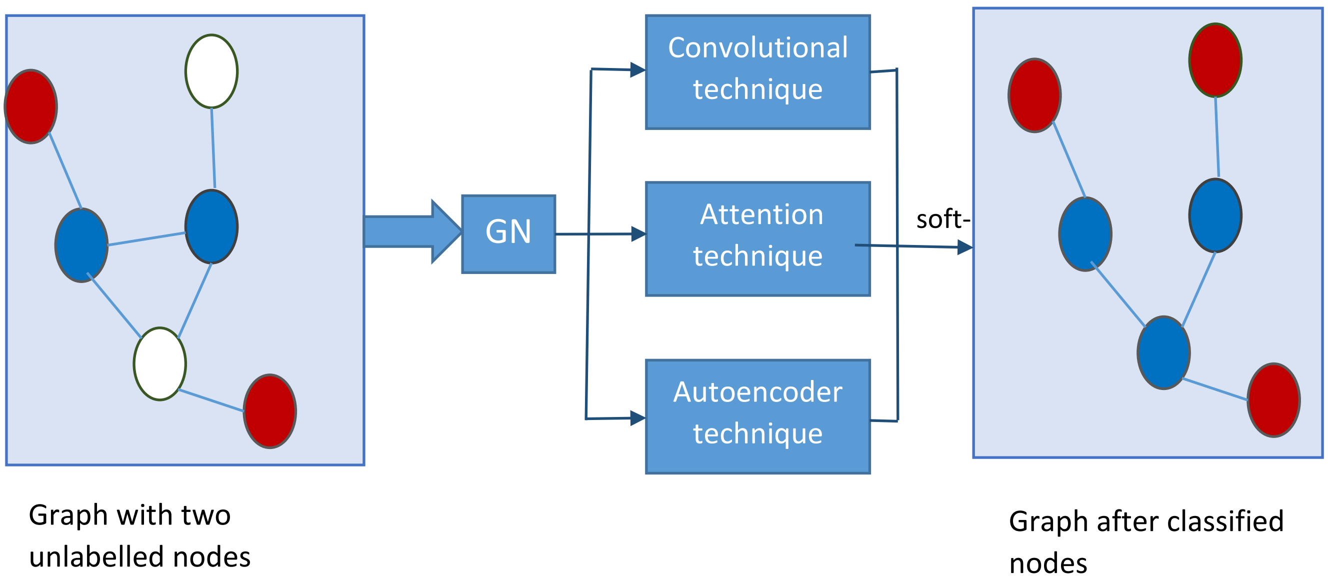
Keywords

Graph Neural Network Node Classification
Graph Convolutional Network
Graph Attention Network

ABSTRACT

Convolutional and recurrent neural networks have been found to be beneficial in enhancing numerous of machine-learning tasks. However, all of the inputs that these deep learning models use, such text or images, are of the Euclidean structure type. Since graphs are a non-Euclidean structure in the machine learning area, it is challenging to apply these neural networks directly to graph-based applications like node classification. Due to increased research focus, graph neural networks—which are created to handle specific graph-based input—have made significant advancements. In this article, we present an in-depth review of the use of graph neural networks for the node classification problem. The recent techniques are first described and broken down into three primary groups: attention technique, convolutional technique, and autoencoder technique. The performance of several approaches is then compared to in-depth comparative tests on a number of benchmark datasets.

Graphical abstract



∗ *Corresponding author*
E-mail address: *asmaadaoud.1959@azhar.edu.eg*

DOI: 10.21608/IJTAR.2024.219355.1068

1. Introduction

The performance of neural networks has greatly improved in many sectors due to the quick development of computational resources and trainable data. The study of graph neural networks (GNNs) has advanced significantly in recent years. Notably, a wide range of GNN designs, such as GCN [1], graph attention networks (GAT) [2] and GraphSAGE [3] have been developed. Then, these designs are used in numerous fields, such as social networks [4], chemistry [5], and biology [6]. Additionally, there has been an increasing trend to add more layers to the models, making them deeper, to increase their expressiveness [7].

In the Euclidean domain, which includes audio, text, and image, deep learning systems have shown significant success [8]. Graph data, one of the common structures that are not Euclidean in the machine learning industry has the characteristics of unknown size, complex topological structure, and always having a variable node ordering [9]. In order to structure data, it is necessary to directly use a common learning process (such as pooling or convolutional operations). However, due to their exceptional ability to represent objects and relationships in a variety of fields, such as community recognition [10], traffic flow prediction [11], and knowledge graphs [12], graph data are crucial structures in the machine learning area. More researchers are dedicating more time to generalizing these successful neural networks to graph analysis [13]. Consequently, GNNs (graph neural networks) have risen in popularity and made a number of advances [14].

GNNs have gained popularity in recent times for a number of graph analysis activities, such as node-focused activities (such as node classification and link prediction) and graph-focused activities (such as graph similarity classification and detection) [15]. Due to the numerous application possibilities, one of the most prevalent types of study in graph analysis is node classification. The goal of the node classification challenge is to assign, using graph information, a specific label for each unlabeled node in the graph [16]. Node classification, for instance, may predict the study subject that every article in the citation networks belongs to [17]. Each node in the protein-protein interaction network can have one or more gene ontology types ascribed to it [18]. Only a small subset of the nodes in the training dataset contain labels, which is the goal of semisupervised node classification, as shown in Figure 1.

We give a study of graph neural networks in order to compare various techniques in node classification. The following list illustrates the paper's contributions:

- This survey offers an extensive review of the current node classification graph neural network models. It shows many well-known algorithms for each category and develops a new taxonomy for these models.
- Based on a thorough evaluation, many popular algorithms from each category are compared. These algorithms are specifically rerunning on a number of well-known benchmark datasets. The results of evaluations are also used to conduct an analysis.

The rest of the paper is structured as follows: Section 2 discusses the first definition of a few notations that are frequently used after introducing certain notions pertaining to node classification. Then, in Section 3, many graph neural network methods of various categories are presented. In Section 4, we evaluate graph neural network methods for node classification on various datasets and analyze these results.

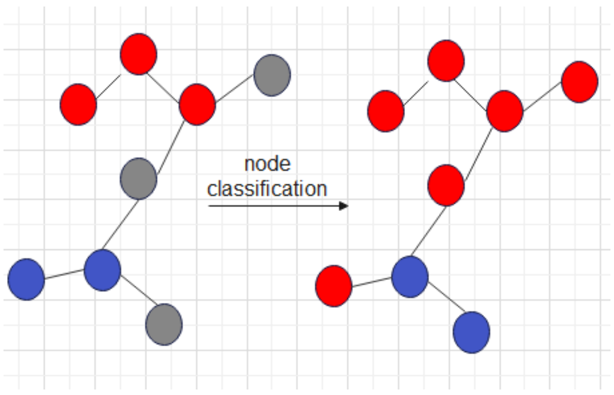


Figure 1. Semi-supervised node classification illustration. Grey relates to unlabeled nodes, while blue and red indicate nodes for which the label is already known. The goal is to label each grey node in accordance with all the information of those colorful nodes.

2 Notations and Definitions

Definition 1 (Graph): Assume that there are n nodes and m edges in the undirected graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is a collection of nodes. The graph is shown by an adjacency matrix denoted by the letter $A \in \{0,1\}^{n \times n}$. If an edge connects nodes v_i and v_j, each element A_{ij} is set to 1; otherwise, it is set to 0. When self-loops are included in the graph, an adjacency matrix known as $A = A + I$ is created. Each node has a d-dimensional feature vector, and the feature matrix of all nodes is displayed by the notation $X \in R^{n \times d}$ where $X = \{X_1, X_2, \ldots, X_n\}$ where X_i is the matching feature vector of node v_i. Additionally, one-hot encoding is utilized as a feature for every node in unattributed graphs, i.e., X = I.

Definition 2 (the *k*-hop neighbors)

The term "k-hop neighbors of u_i" refers to a group of nodes that are actually k hops away from u_i and is formulated as

$$N_k(i) = \{u_j \mid i \neq j, \min(sp(i, j), K) = k, \forall u_j \in U\}. \quad (1)$$

When there is no edge between u_i and u_j, the shortest path, denoted by sp(i, j), will be infinite. K is the highest possible number of hops.

3 Categorization and Frameworks

This section provides a thorough description of different graph neural networks that can be applied to node classification. Convolutional graph neural networks, attention graph neural networks, and graph autoencoders are the categories of these graph neural networks. We provide a brief summary of each category in the paragraphs that follow.

3.1 Graph Convolutional Technique

One of the most often used information aggregation techniques in graph analysis is the graph convolutional mechanism. This mechanism's fundamental principle is to use pooling or convolutional operations on the graph structure to obtain a higher representation of each node, which is subsequently used in the node classifier. Graph convolutional networks (GCNs), which are distinct from CNNs on images and based on GNN models, are not affected by the arrangement of nodes.

3.1.1 ChebNet Graph Convolutional Network

Defferrard et al. [19] developed a spectral-based graph convolutional network named ChebNet, which includes a quick localized spectral graph filter built from the Chebyshev polynomial, in order to generalize CNN operators to the graph domain. Specifically, ChebNet consists of three primary processes, the construction of localized convolutional filters, reducing the size of a graph, and a pooling operation of the graph.

Graph Laplacian, which has the definition $L = D - A \epsilon R^{n\times n}$, is an essential operator in spectral graph analysis [20]. L also has a normalized form that is created as

$$L = I_n - D^{-\frac{1}{2}} A \; D^{-\frac{1}{2}} \tag{2}$$

The formula for the Laplacian is $L = U\Lambda U^T$, where is the diagonal matrix containing the L-derived eigenvalues, and U stands for the Fourier basis. diag() is a diagonal matrix given by the input vector or matrix.

The input signal $x \in R^n$ (each element associated with a node) is then filtered using a spectral filter g_θ , which is illustrated as follows:

$$g_\theta(L)x = g_\theta(\mathrm{U}\Lambda U^T)x = Ug_\theta(\Lambda)U^T x \tag{3}$$

Where $g_\theta(\Lambda) = diag(\theta)$ is a nonparametric filter, θ refers to the Fourier coefficients, and $U^T x$ refer to the graph Fourier transform of **x**.

The nonparametric filter g_θ, however, is difficult to learn and unable to localize in space. Defferrard et al. [19] use a recursive formulation of Chebyshev to compute $g_\theta(L)$ and a polynomial filter to solve these issues. The filter can therefore be parametrized as

$$g_\theta(\Lambda) = \textstyle\sum_{k=0}^{K-1} \theta_k T_k(\widetilde{\Lambda}) \tag{4}$$

where $[-1,1]$, λ_{max} represents the greatest eigenvalue of $L, \widetilde{\Lambda} = 2\Lambda/\lambda_{max} - I_n$ represents a diagonal matrix with all components in the range [1, 1], and the parameter $\theta \in R^k$ signifies all coefficients of the Chebyshev polynomial $T_k(x)$. $T_k(x)$ can be calculated precisely using a recursive method. $T_0(x) = 1, T_1(x) = x$, respectively. $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$. The previous equation can be reformatted as follows:

$$g_\theta(\mathrm{L})x = \textstyle\sum_{k=0}^{K-1} \theta_k T_k(\,L\,)x \tag{5}$$

The k-th order Chebyshev polynomial $T_k(\tilde{L})$ Rn n can be evaluated using the scaled graph Laplacian $= 2L/\lambda_{max} - I_n$. Since Eq. (5) is a K-order polynomial in the Laplacian, the central vertex is therefore dependent on its K-hop neighbors.

The hidden state is recursively updated as $\bar{x}_k = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2}$, where $\bar{x}_0 = x$ and $\bar{x}_1 = \tilde{L}x$, according to Defferrard et al.'s [19] denotation of $\bar{x}_k = T_k(\tilde{L})x \in R^n$. This iterative procedure is depicted. The full filter, which involves $\mathcal{O}(K_m)$operations, can be written as $z = g_\theta(L)x = [x_0, x_1, \dots., x_{k-1}]\theta$.

3.1.2 Graph Convolution Networks (GCN)

Kipf and Welling [21] also suggest a spectral based on GCN that aggregates the feature vector associated with each node of its first-order approximate neighbors [22], an alternative to using the information originated from K-hop neighbors to represent the ChebNet node [19]. The last hidden representation associated with each node is then obtained by a deep neural network architecture that is composed of stacking the graph convolutional layers several times. As a result, the obtained representation is similar to ChebNet in that it also contains information about its multi-hop neighbors [19].

Kipf and Welling [21] specifically defined that amount of hops as $K = 1$. As a result, Eq. (5) is transformed into a linear function and written as

$$z = \theta_0 T_0(\tilde{L})x + \theta_1 T_1(\tilde{L})x = \theta_0 x + \theta_1\left(\frac{2}{\lambda_{max}} L - I_n\right)x \tag{6}$$

Kipf and Welling [21] addressed the issue of overfitting to a graph's local structure and making the greatest eigenvalue as a λ_{max}=2 to reduce the number of operations and using a singular parameter $\theta = \theta_{0=-}\theta_1$. With this configuration and Eq. (2), (6) is reformed as

$$z = \theta\left(I_n + D^{-\frac{1}{2}} A \, D^{-\frac{1}{2}}\right)x \tag{7}$$

Here, all of the eigenvalues for the formula fall between [0, 2], and all layers share the filter's parameters.

Be aware that stacking such convolutional operators could lead to issues like numerical instability and exploding or vanishing gradients when creating a deep neural network model. Kipf and Welling [21] employed a renormalization trick technique to address these issues, so the expression of Eq. (7) became

$$I_n + D^{-\frac{1}{2}} A \, D^{-\frac{1}{2}} \rightarrow \widetilde{D}^{-\frac{1}{2}} \tilde{A} \, D^{-\frac{1}{2}} \tag{8}$$

where $\tilde{A} = A + I_n$ is the adjacency matrix with a self-loops.

The definition of Eq. (8), which is used when the input signals $x \in R^n$ with just one channel, was then generalized by Kipf and Welling [21] to the case when each signal has multiple channels $X \in R^{n \times d}$. Here, "d" stands for the dimension of the node feature vector, or the quantity of input channels. The following is the definition of the convolutional filter for signal X:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} D^{-\frac{1}{2}} \text{XW} \tag{9}$$

Here, the filter parameter matrix $W \in R^{d \times f}$, the convolved feature matrix $Z \in R^{n \times f}$, and the dimension of the embedding feature f are all present.

A multi-layer Graph Convolution Network with a layer-wise propagation rule is what Kipf and Welling [21] intend to construct after designing the convolutional filters of each layer:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} D^{-\frac{1}{2}} H^l W^l) \tag{10}$$

where $H^{(l)} \in R^{n \times h}$ is the matrix of hidden layers, $W^{(l)}$ is a trainable weighted matrix, and h is the upper representation dimension. The input signal X is used to initialize the variable $H^{(0)}$.

Kipf and Welling [21] suggested a two-layer GCN model for semi-supervised node classification. Then Kipf and Welling [21] established the following forward propagation model:

$$Z = f(X, A) = \sigma(\hat{A}\, ReLU(\hat{A} X W^{(0)}) W^{(1)} \tag{11}$$

Where $W^{(0)}$ is a matrix that maps to the hidden representation of the input feature, $W^{(1)}$ is a matrix that maps the hidden representation to the output, and $\sigma(.)$ Is a softmax function.

3.1.3 GraphSAGE Convolutional Network

ChebNet and GCN, however, cannot generalize to previously unseen nodes because they are fundamentally transductive and depend on all nodes being present throughout the training phase. Hamilton et al. [23] proposed a spatial based on GCN called Graph-SAGE. To obtain a higher representation for each node, GraphSAGE combined the feature information of nodes and the structural features of a local neighborhood of the node. This enables a model to become inductive and deal with unseen nodes. Hamilton et al.'s [23] set of aggregators was created to learn embeddings by aggregating the data of surrounding nodes around the current central node rather than training several hidden representations for each node independently. These aggregators are then used to create GraphSAGE's forward propagation algorithm. Existing K aggregators can be identified by the symbols AGGREGATEk, $\forall k \in \{1, \dots, K\}$, and K parameter matrices W^k, $\forall k \in \{1, \dots, K\}$, which serve as a converters between various hops. The LSTM aggregator, mean aggregator and pooling aggregator are the three types of aggregators.

Hamilton et al. [23] first used AGGREGATE to generate the neighborhood vector aggregated $h^k_{N(i)}$ by using the information of all neighbors of v_i which was generated in the preceding time step. This allowed each step k to retrieve the hidden layer h^k_i of each target node v_i. The target node v_i's current state is created by concatenating $h^k_{N(i)}$ with its before hidden state, h^{k-1}_i, and then using W^k's activation function to transform this concatenated vector. Each node's final feature representation Z_i is constructed in the K-th phase by repeating the previously mentioned procedure.

3.1.4 Graph Neural Network Using Feature Selection-Based Centrality Measures (GNNFC)

Previous methods used all features to represent nodes. It becomes more computationally expensive as the number of hops rises since the number of combinations for input features increases exponentially. In addition, it is necessary to enhance the model's prediction capability. As a result, the feature selection approach may be used to build a GNN model. In [24], GNNFC learns to detect relevant features while minimizing the influence of insignificant features by first taking all features as input.

The Chi-square between the objective and each feature is computed in GNNFC, and the features that have the highest Chi-square scores are selected. Since graph centralities have been used to capture important information from the graphs, with regard to graph centrality measures, we provide additional measurements such as betweenness and closeness. These features are computed, combined with the chosen features derived from the input features, and then fed into GNN.

The merge procedure can be defined as follows:

X=Concat(selectedFeat,CentralityFeat) (12)

where selectedFeat denotes the features chosen using the Chi-square approach and centralityFeat denotes the features computed using the degree, betweenness, closeness, and eigenvector centrality measures.

Then, Asmaa et al. [24] proposed a design space for GNNs composed of the following four steps:

- A preprocessing layer that creates initial node representations using a multilayer perceptron (MLP).

- A GCN-based message-passing layer.

- A post-processing layer that creates final node embeddings using MLP.

- The final node embeddings feed into a SoftMax layer for predicting the node class.

The original node representation is processed $h^0_v = X_0$ using an MLP in the first phase to create a message. The representation of a node is then repeatedly updated in a message-passing step by aggregating the neighbors' representations. After doing aggregation several times of k, a node's representation captures the structural data in its k-hop network neighborhood.

$$h_v^{(k+1)} = AGG\left(\left\{ACT\left(GN\left(W^{(k)}h_u^{(k)} + b^{(k)}\right)\right), u \in N(v)\right\}\right) \quad (13)$$

Where $h_u^{(k)}$ is the $k-th$ layer embedding of node, $W^{(k)}, b^{(k)}$ are trainable weights, and $N(v)$ is the local neighborhood of v.

The accepted GNN in (13) starts with a linear layer and then has a number of functions, including the nonlinear activation function ACT $(.)$, which takes $ReLU$ into account, batch renormalization (BR), and aggregation function (AGG), which takes SUM into account.

We used GN to normalize node features in Eq. (13) while training. Hamilton et al. claim that jobs where node feature information is much more essential than structural information or where there is a wide of node degrees range are the ones where normalization is most helpful. The GN is particularly effective at stabilizing the value in GNNs, especially deep GNNs. As a result, normalization is being used by more and more GNNs.

Deep GNN models typically produce poor performance as a result. In this case, we examine dense connections termed SKIP-CAT, which combines embeddings from all prior layers. One method is to use the skip connections to concatenate the outputs and inputs of GNN.

3.1.5 Dual-Net Graph Neural Network

A Dual-Net GNN architecture that consists of a classifier model and a selection model is suggested by Sunil et al. in [25]. The selector model learns to give the classifier the ideal input subset for the greatest performance, while the classifier model trains on a subset of the input node's attributes to predict node labels. Together, these two models are trained to identify the most accurate collection of features for node label predictions.

Classifier Network:

It is a two-layered MLP neural network. The parameter θ in $f_c(\theta; X, m)$ is used to parameterize the classifier network. A portion of the node feature matrices, $\in M$, serves as the network's input. Each input matrix is linearly processed, and the result is summed in the first layer. The second layer is then mapped to non-linearity (ReLU). After the joint model has been trained, the formula $f_c(\theta^*; X, m^*)$ is applied to the test dataset, where m^* is the ideal subset of node feature matrices as input and indicates the learned classifier parameters.

Selector Network:

The input/output configuration of the selection network differs from that of the other two-layered MLPs. Based on the classifier's performance history, the selector's job is to identify the best feature subset of X. The selector model with φ parameter that is used, denoted as $f_c(\varphi, m)$ .A one-hot encoding vector $\vec{m}$ of size $2K + 1$ that represents the subset of feature matrices to be chosen in accordance with m serves as the input to the selector network. For instance, the subset $\{X, (A+I)X, A^3X\}$ has $\vec{m} = (1,0,1,0,0,1,0)$ for $K = 3$. One scalar value is produced as the output. The selector net's goal is to develop the ability to predict the classifier's typical performance on the mask matching the input subset.

3.1.6 Feature Extraction and Selection for GNN

Deepak et al. [26] Considered a network with 'n' nodes and 'f' features. By using the notion of feature selection, we can reduce the number of features from f to k where k is the number of features picked (where $k < f$). The dataset is then trained using the Gumbel feature selection matrix, which is the matrix with the features chosen when Gumbel-Softmax is applied, and its accuracy is evaluated. They used the Adam optimizer as a metric for optimization and negative log-likelihood loss as a loss function to determine the loss.

As employed in their experiment, the two-layer Graph Convolution Network (GCN) is defined as

$$GCN(X, A) = Softmax(A(ReLu(AXW_GW_1))W_2) \quad (14)$$

We employ the following two-layer Graph Convolution Network, which is defined below, to validate the chosen features and determine the accuracy for classification.

$$GCN(X, A) = Softmax(A(ReLu(AXW_G'))W_2) \quad (15)$$

Where W_G: Gumbel-Softmax feature extraction/feature selection matrix.
W_G': feature extraction/feature selection matrix.
W_1, W_2: Layer-specific trainable weight matrix.

3.2 Attention Technique

One of the most practical artificial intelligence architectures is the attention mechanism. Different neighbours should contribute differently to the target node rather than using a consistent weight. Additionally, every node has a different number of neighbors. Dealing with varying sizes of inputs and concentrating on the most important information are advantages of attention mechanisms. Applying an attention mechanism to node classification is thus natural.

3.2.1 Graph Attention Networks (GAT)

When creating the higher hidden representation, Velickovi'c et al. [3] proposed an attention-based GCN that may be utilized to build arbitrary deep GAT by stacking such attention layers. This allowed them to capture the varying importance of the neighborhood for a target node. The GAT model's overall goal is to create a new set of features. Using $X = \{X_1, X_2, \ldots .X_n\}$ as the model inputs, $Z = \{Z_1, Z_2, \ldots .Z_n\} \epsilon R^{n \times f}$ for all nodes. To obtain the hidden representations $H_i = WX_i \in R^f$ for each node, GAT initially uses a learnable linear transformer. This transformer's parametrized weight matrix is indicated here by the notation $W \in R^{f \times d}$. Following that, Velickovic et al. [3] developed an attentional function to calculate attention weights between each node pair using the previously mentioned hidden representation

$$e_{ij} = ATTENTION\left(H_i, H_j\right) \tag{16}$$

In more detail, e_{ij} indicates the weighting of the node v_j in the representation of a node v_i. Keep in mind that in Eq. (16), Velickovi'c et al. [57] only employ the first-order relevant neighbors. Using a softmax function, e_{ij} is normalized to improve the comparability of the weight coefficients between various nodes, and the improved version is described as

$$\alpha_{ij} = softmax_j\left(e_{ij}\right) = \frac{\exp\left(e_{ij}\right)}{\sum_{vk \in N(i)} \exp\left(e_{ik}\right)} \tag{17}$$

In reality, Velickovi'c et al. [3] computed the normalized coefficients using simply a single FNN (forward neural network) with an activation function. Eq. (17) is reformed as a result as

$$\alpha_{ij} = \frac{\exp\left(\sigma\left(a^T\left[H_i \| H_j\right]\right)\right)}{\sum_{vk \in N(i)} exp\left(\sigma\left(a^T\left[H_i \| H_k\right]\right)\right)} \tag{18}$$

where $\sigma(.)$ is implemented by the function LeakyReLU and $a \in R^{2f}$ is the weight vector of the suggested attention mechanism carried out by a single-layer FNN. Once all of a node's neighbors' coefficients have been calculated, all of their neighbors are combined linearly to produce the last representation for each target node, which is displayed as follows:

$$Z_i = \sigma\left(\sum_{v_j \in N(i)} \alpha_{ij} H_j\right) \tag{19}$$

3.3 Graph Autoencoders Technique

Most unsupervised technologies now for learning a low-dimensional embedding from massive unlabeled training data is the autoencoder mechanism. Additionally, there are numerous node classification data sets with a few nodes that have labels. This approach must be used in order to learn a higher representation for each node.

3.3.1 Deep Graph Infomax (DGI)

Most unsupervised learning techniques now in use for creating node embeddings are built on the random walk technique. These methods, however, focus excessively on localized structure information [28], and the performance largely depends on the selection of hyperparameters [29]. DGI is an unsupervised learning technique, which is based on the mutual information between the entire graph's global representation and the patch representation of special input, was developed by Velickovic et al. [30] as a solution to this issue. Particularly, DGI introduces mutual information maximization into the graph data.

Velickovi'c et al. [30] used a graph convolutional encoder: $R^{n \times d} \times R^{n \times n} \rightarrow R^{n \times f}$, such that $= f(X, A)$, to construct H_i (a high-level embedding) $\forall v_i$. Keep in mind that H_i was referred to by Velickovi'c et al. [30] as the patch representation of v_i where this embedding was created by summing up a patch of graph data. A summary vector s was developed by Velickovic et al. [30] to represent the global data of the complete network. The readout function $\mathcal{R} : R^{n \times d} \rightarrow R^d$, which accepts the derived patch representations as input and may be written as $s = \mathcal{R}\left(f(X, A)\right) = \mathcal{R}(H)$, is used to compute the summary vector s.
Then, Velickovi'c et al. [30] used the discriminator $\mathcal{D} : R^d \times R^d \rightarrow R$ to assign a probability score to each patch-summary pair. Keep in mind that the objective function will use this score after that.

4 Evaluation Analysis

On a number of well-known node classification datasets, we compare the aforementioned graph neural network models in this section. We begin by thoroughly describing the statistical data and parameter settings of the datasets. Finally, classification outcomes from the various approaches are offered.

4.1 Datasets

With two benchmark datasets, as mentioned in Table 1, that are often used in GNN literature, we perform an evaluation of the fully supervised node classification problem. The citation network-based datasets Cora and Citeseer [17] are generally regarded as homophily datasets. We use publicly available data splits to give a fair comparison. Since all parameters for all algorithms are specified based on the original articles, we compare the performance of different approaches using accuracy evaluation criteria, as in [31]

Table 1. Details surrounding the datasets used in this paper

Datasets	Cora	Citeseer
Number of Nodes	2708 node	3327 node
Number of Edges	5429 edge	4732 edge
Number of Avg. Deg	3.90	2.74
Number of Features	1433	3703
Number of Classes	7	6

4.2 Evaluation results

This section discusses the thorough evaluation of the existing GNN model for node classification tasks. The outcomes are also extensively reported as shown in Table 2. Two citation network benchmark datasets, which are common citation networks with identical training and testing split [17], are selected to compare the performance of various techniques.

Table 2. compares the accuracy of the node classification results for the well-known GNN models.

	Cora	Citeseer
ChebNet [32]	78.08± 0.86	67.87 ± 1.49
GCN [24]	87.28 ± 1.26	76.68 ± 1.64
GraphSAGE [24]	86.90 ± 1.04	76.04 ±1.30
GNNFC [24]	89.5	80.1
Dual-Net GNN [25]	87.77 ±1.16	77.15 ± 1.58
Deepak's method [26]	73.80	61.80
GAT [33]	82.68 ± 1.80	75.46 ± 1.72
DGI [26][30]	82.30 ± 0.6	71.80 ± 0.7

5 Conclusion

We gave an overview of graph neural networks in this research and compared them in node classification tasks. These graph analysis algorithms were categorized into three groups based on the three main learning paradigms: convolutional technique, attention technique, and auto-encoder technique. For each category, a thorough introduction to a number of well-liked techniques was given. To examine the effectiveness of node classification, detailed comparative tests on two benchmark datasets were also run.

References

[1] T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks, *Prepr. arXiv1609.02907*, 2016. https://arxiv.org/abs/1609.02907

[2] W. L. Hamilton, R. Ying, and J. Leskovec, Inductive representation learning on large graphs, *Prepr. arXiv1706.02216*, 2017. https://arxiv.org/abs/1706.02216

[3] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, Graph attention networks, *Prepr. arXiv1710.10903*, 2017 https://arxiv.org/abs/1710.10903

[4] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, Graph convolutional neural networks for web-scale recommender systems, Proceedings of the 24th ACM SIGKD, Jun. 2018, doi: 10.1145/3219819.3219890. https://arxiv.org/abs/1806.01973

[5] W. Jin, R. Barzilay, and T. Jaakkola, Junction tree variational autoencoder for molecular graph generation, in Proceedings of the 35th International Conference on Machine Learning, 80(2018) 2323–2332. https://arxiv.org/abs/1802.04364

[6] M. Zitnik and J. Leskovec, Predicting multicellular function through multi-layer tissue networks, Bioinformatics, 33(14)(2017) 190-198, doi: 10.1093/bioinformatics/btx252. https://academic.oup.com/bioinformatics/article/33/14/i190/3953967

[7] Y. Rong, W. Huang, T. Xu, and J. Huang, DropEdge: Towards deep graph convolutional networks on node classification, *Prepr. arXiv1907.10903*, 2019. https://arxiv.org/abs/1907.10903

[8]L. Elman, Finding structure in time. Cogn, Sci. **14**(2)(1990) 179–211. https://www.sciencedirect.com/science/article/abs/pii/036402139090002E

[9] E. Bullmore, O. Sporns, Complex brain networks: graph theoretical analysis of structural and functional systems, Nat. Rev. Neurosci. **10**(3) 186 (2009). https://www.nature.com/articles/nrn2575

[10]J. Yang, J. Leskovec, Community-affiliation graph model for overlapping network community detection, In: Proceedings of 12th IEEE International Conference on Data Mining,(2012) 1170–1175. https://cs.stanford.edu/people/jure/pubs/agmfit-icdm12.pdf

[11] W. Huang., G. Song, H. Hong, K. Xie, Deep architecture for traffic flow prediction: Deep belief networks with multitask learning. IEEE Trans. Intell. Transp. Syst. 15(5) (2014) 2191–2201. https://ieeexplore.ieee.org/abstract/document/6786503

[12] G. Ji, K. Liu, S. He, J. Zhao, Knowledge graph completion with adaptive sparse transfer matrix, In: Proceedings of the 30th AAAI Conference on Artificial Intelligence,(2016) 985–991. https://ojs.aaai.org/index.php/AAAI/article/view/10089

[13] M. Gori, G. Monfardini, F. Scarselli, A new model for learning in graph domains, In: Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, (2005) 729–734. https://ieeexplore.ieee.org/document/1555942

[14] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw, **20**(1) (2009) 61–80. https://ieeexplore.ieee.org/document/4700287

[15] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, D. Song, Neural network-based graph embedding for cross-platform binary code detection. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, (2017) 363–376. https://arxiv.org/abs/1708.06525

[16] P. Kazienko, T. Kajdanowicz, Label-dependent node classification in the network, Neurocomputing. **75**(1)(2012) 199–209. https://www.sciencedirect.com/science/article/abs/pii/S092523121100508X

[17] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, Eliassi- Rad, Collective classification in network data, AI Mag. **29**(3), 93–93 (2008) https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2157

[18] A. Subramanian, P. Tamayo, V.K. Mootha, S. Mukherjee, B.L. Ebert, M.A. Gillette, Paulovich, A., Pomeroy, S.L., Golub, T.R., Lander, E.S., et al, Gene set enrichment analysis: a knowledge based approach for interpreting genome-wide expression profiles. Proc. Natl. Acad. Sci USA **102**(43) (2005). 15545 15550. https://pubmed.ncbi.nlm.nih.gov/16199517/

[19] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, In: Proceedings of the 30th Conference on Neural Information Processing Systems, (2016) 3844–3852. https://arxiv.org/abs/1606.09375
[20] D.A. Spielman, Spectral graph theory and its applications, In: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science,(2007) 29–38.
https://ieeexplore.ieee.org/document/4389477
[21] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, In: Proceedings of the 4th International Conference on Learning Representations (2016) https://arxiv.org/abs/1609.02907
[22] D.K. Hammond, P. Vandergheynst, Gribonval, R, Wavelets on graphs via spectral graph theory. Appl. Comput. Harmon. Anal. **30**(2) (2011), 129–150. https://arxiv.org/abs/0912.3848
[23] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, In: Proceedings of the 31st Conference on Neural Information Processing Systems, (2017) 1024–1034.
https://arxiv.org/abs/1706.02216
[24] A. M. Mahmoud, A.S. Desuky, H. F. Eid and H. A. Ali, Node classification with graph neural network based centrality measures and feature selection, International Journal of Electrical and Computer Engineering (IJECE), 13(2)(2023) 2114-2122. https://ijece.iaescore.com/index.php/IJECE/article/view/28677
[25] K.M. Sunil, Feature selection: Key to enhance node classification with graph neural networks, CAAI Transactions on Intelligence Technology, 8(2023) 14-28. https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/cit2.12166
[26] D. B. Acharya and H. Zhang, Feature selection and extraction for graph neural networks, in *Proceedings of the 2020 ACM Southeast Conference*, (2020) 252–255, doi: 10.1145/3374135.3385309. https://arxiv.org/abs/1910.10682
[27] https://petar-v.com/GAT/.
[28] L.F. Ribeiro, P.H. Saverese, D.R. Figueiredo, Struc2vec: Learning node representations from structural identity, In: Proceedings of 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (2017) 385–394.
https://arxiv.org/abs/1704.03165
[29] A. Grover, J. Leskovec, Node2vec: Scalable feature learning for networks, In: Proceedings of the 22ndACMSIGKDDInternational Conference on Knowledge Discovery and Data Mining,(2016) 855–864. https://arxiv.org/abs/1607.00653
[30] P. Veliˇckovi´c, W. Fedus, W.L. Hamilton, P. Liò, Y. Bengio, R.D. Hjelm, Deep graph infomax, In: Proceedings of the 6th International Conference on Learning Representations (2018). https://arxiv.org/abs/1809.10341
[31] S. Xiao, S. Wang, Y. Dai et al, Graph neural networks in node classification: survey and evaluation, Machine Vision and Applications 33(4) (2022). https://doi.org/10.1007/s00138-021-01251-0.
[32] M. He, Z. Wei, and J. Wen, Convolutional Neural Networks on Graphs with Chebyshev Approximation, Revisited, 36th Conference on Neural Information Processing Systems (NeurIPS), 2022. https://arxiv.org/abs/2202.03580
[33] S. K. Maurya, X. Liu, and T. Murata, Simplifying approach to node classification in Graph Neural Networks, Journal of Computational Science, 62(2022), doi: 10.1016/j.jocs.2022.101695.
https://arxiv.org/abs/2111.06748