GAT-Edge: Graph Attention Neural Network with Adjacent Edge Features

Anonymous ACL submission

Abstract

Edge features are a crucial component of graph data as they provide a wealth of information that can enhance model performance. In this paper, we propose an improved model called GAT-Edge, which builds upon the graph attention network by optimizing the attention mechanism to incorporate edge feature information. By leveraging adjacent edge features in the graph, our model can assist downstream tasks such as node classification. The connection between nodes in graph data is often enriched by the adjacent edges, which provide more effective and abundant information. To exploit this, our model combines edge features and node features in the attention calculation, convolving them together to generate new attention coefficients. This approach facilitates efficient information transmission and aggregation between nodes, leading to improved performance. We apply our new model to several citation networks commonly used in the field of graph neural networks for node classification, and compare it with the current mainstream graph convolution neural network models. Our results demonstrate that our model achieved better accuracy, highlighting the importance and research value of mining adjacent edge features in graphs.

1 Introduction

A series of deep learning models represented by fully connected neural networks, convolutional neural networks (Lecun et al., 1998) and generative additive networks (Goodfellow et al., 2014) help us explore the information hidden behind the data, so to better understand the world. However, most of the popular deep learning models are strictly defined on structure data, which is difficult to be directly extended to non-structure data such as graph data. Therefore, graph convolution neural network (GCN) (Kipf and Welling, 2017), graph attention network (GAT) (Veličković et al., 2018), GraphSage (Hamilton et al., 2017) and other methods have been proposed, effectively expanding the application of deep learning in graph data.

Graph data typically consists of nodes and edges, where nodes represent real-world entities and adjacent edges represent their interactions or relationships. For instance, social networks can be modeled as graphs, with users represented as nodes and their relationships as edges. Each node is typically associated with a multi-dimensional feature vector, such as user data, to capture its characteristics. Adjacent edges in a graph can be represented by eigenvectors to capture communication, interaction, and other behaviors among (n) users. Graph data is irregular in structure, making traditional machine learning methods unsuitable. However, with advances in deep learning and computing power, graph neural networks (GNNs) have become a popular and efficient method for processing graph data.

GNNs extend the methods of local receptive fields, weight sharing, and spatial downsampling from Euclidean structure data, such as images, to non-Euclidean structured data, such as graphs. The self-attention mechanism used in the transformer model in natural language processing has been adapted for information aggregation between nodes, leading to the development of the graph attention network (GAT). GAT is based on the idea of a convolutional neural network and uses the Laplace matrix to complete the definition of spectral domain transformation on a graph. In contrast to the graph convolutional neural network (GCN), which convolutes using the Laplacian matrix, GAT aggregates node information using the attention mechanism. This allows GAT to fuse information from both node features and graph topology, making it more effective in information aggregation. Moreover, GAT's adaptability enables it to effectively capture correlations between node content and information aggregation.

To solve above problems, we propose an improved model called GAT-Edge, which builds upon the graph attention network by optimizing the attention mechanism to incorporate edge feature information. Our contributions can be summarized as follows:

- The paper improves the original attention mechanism of GAT by incorporating edge feature information to address the lack of structural information processing.
- To introduce edge features, the paper splices them into two node features and increases the dimension of the mapping matrix to calculate a new attention coefficient.
- Using the improved attention mechanism, each neural network hidden layer calculates multi-head attention, summarizes the information according to the structure of GAT, and finally classifies nodes.
- The paper manually constructs a three-dimensional feature vector from directed edges, experimental results demonstrate that the GAT-Edge model significantly improves the accuracy index of node classification and highlights the importance of exploring edge features in graph data.

2 Rwlated Work

2.1 Graph Neural Network

GNN(Scarselli et al., 2009), initially proposed by Scarselli et al. (2009) , connects node features through a latent function (Thang et al., 2022). Wu (Wu et al., 2021) categorize GNN models into four groups: cyclic GNNs, Graph Convolutional Neural Networks, graph Autoencoders, and Spatio-Temporal Graph Convolutional Networks. GCN has spectral and spatial modes. Spectral CNN has limitations like eigenvalue sensitivity and topology specificity (Denton et al., 2014). Subsequent models, e.g., Chebnetp (Hammond et al., 2011) and AGCN (Li et al., 2018), simplify computations and enhance relationships between nodes.

2.2 Attention Mechanism

The self-attention mechanism(Bahdanau et al., 2016), pioneered by Bahdanau et al. (2016) for Seq2Seq models in machine translation, addresses limitations in processing long sequences. The mechanism computes attention to each hidden layer, mitigating gradient vanishing. The transformer model (Vaswani et al., 2017) popularized self-attention in NLP and extended it to computer vision, aiding image understanding and text-image interactions (Xu et al., 2015). In graph data, GAT introduced self-attention to enhance node communication (Wei et al., 2022). An efficient method for spectral-based information filtering was proposed (Vaswani et al., 2017). Multi-head attention improves attention stability and learning (Vaswani et al., 2017). This mechanism helps the model filter graph noise and improve signal-to-noise ratio (Lee et al., 2018).

Symbol	Description
$N(v)$	Neighbor set of node v
n	The total number of nodes in the graph data
m	The total number of adjacent edges in the graph data
A	Adjacency matrix of graph data
d	The dimension of node feature vectors on a graph
c	Dimension of adjacent edge feature vectors on a graph
$\mathbf{X} \in R^{n\times d}$	The characteristic matrix of nodes on a graph
$\mathbf{x}_v \in R^d$	The feature vectors of node v on the graph
$\mathbf{X}^e \in R^{m\times c}$	The characteristic matrix of adjacent edges on a graph
$\mathbf{x}^e_{uv} \in R^c$	The feature vectors of adjacent edges between node u and node v on the graph
$\mathbf{H} \in R^{n\times b}$	Node feature matrix of hidden layers
$\mathbf{h}_v \in R^b$	The feature vectors of node v in the hidden layer
K	Number of hidden layers
$\mathbf{W}, \mathbf{W^e}$	Trainable parameters and parameter matrices in graph neural networks

Table 1: Basic statistical information of the three datasets

2.3 Edge Features in GNN

Several methods aim to incorporate edge features in graph data. Message Passing Neural Network (MPNN) (Gilmer et al., 2017) involves two stages, but it doesn't recognize node-edge correlations. Edge-Enhanced Graph Neural Network (EGNN) introduces edge weights into self-attention, but it only uses the original edge features in the first layer, causing information decay with layers (Gong and Cheng, 2019). In summary, they have limitations in recognizing node-edge correlations, introducing edge weights at multiple layers, and handling large directed graphs.

3 GAT-Edge

3.1 GAT

Given a graph $\mathbf{G} = \{\mathbf{V}, \mathbf{E}\}$, where $v_i \in V$ is a node, $i = 1, ..., n$. $e_{ij} \in E$ represents the edge between v_i and v_j. For a directed graph $e_{ij} = 1$ means there is an edge from v_i to v_j, otherwise $e_{ij} = 0$. Table 1 gives the notations used in this article.

GAT introduced the attention mechanism from NLP into GNN, proposing the graph attention network. It uses self-attention to impose varying weights on each first-order neighbor node and employs a mask attention mechanism to focus on the first-order neighbor nodes while constructing a mask matrix to filter out greater-than-second-order interference. Weight allocation is recalculated using the softmax function.

$$\alpha_{ij} = softmax_j(b_{ij}) = \frac{exp(b_{ij})}{\sum_{k \in N_i} exp(b_{ik})} \quad (1)$$

Among them, b_{ij} represents attention coefficient between node v_i and node v_j , N_i represents the set of all first-order neighbor nodes of v_i . When calculating the attention coefficient in GAT, the feature vector x_j of node v_i and the feature vector x_j of node v_j using trainable parameter matrices $\mathbf{W}$ for mapping, respectively. Due to the fact that the attention between two nodes is actually asymmetric, i.e. $e_{ij} \neq e_{ji}$, therefore, concatenate the mapping results of two vectors. Afterwards, a single-layer feedforward neural network $\mathbf{a}^T$ is used to map the concatenated vectors onto real numbers. Finally, using LeakyReLU for activation, the attention coefficient was obtained:

$$b_{ij} = LeakyReLU(\mathbf{a}^T[\mathbf{W}x_i || \mathbf{W}x_j]) \quad (2)$$

Then, the weighted sum of neighbor information is performed on the nodes. The specific details of weighting are as follows:

$$\mathbf{h}_i^{'} = \sigma(\sum_{j \in N_i} \alpha_{ij} \mathbf{W} \mathbf{h}_{ij}) \quad (3)$$

Among them, h_j represents the input vector of node v_j in this layer, while $h_i^{'}$ represents the output vector of node v_i in this layer. σ is the softmax function. However, a single attention mechanism is often not stable, so GAT introduces a multi-head attention mechanism to stabilize the weights between nodes, thereby improving the model's representation ability. For the output vector of the middle hidden layer, GAT uses K independently trained W^k to calculate attention and weight separately. Then concatenate the results calculated by each independent attention head to obtain the final output vector of this layer. Namely:

$$\mathbf{h}_i^{'}(K) = ||_{k=1}^{K} \sigma(\sum_{j \in N_i} \alpha_{ij}^k \mathbf{W}_j^{\mathbf{h}}) \quad (4)$$

3.2 GAT-Edge

GAT excels in various graph deep learning tasks but, like traditional GCN, focuses on node characteristics for convolution, neglecting the importance of adjacent edges. In social networks, capturing interaction information from edges is crucial. By assessing edge information to update node information weights, hidden node representations can be trained more effectively for better downstream task performance. Directed graphs offer more real-world fidelity than undirected ones, but traditional graph convolution networks struggle with directed graphs due to Laplacian matrix symmetry. While GAT works with directed graphs, it aggregates information vertex by vertex without clear edge direction in input features. To address these limitations and incorporate edge features while distinguishing edge directions in directed graphs, this paper proposes an attention mechanism building upon the graph attention network.

3.2.1 Construction of edge features

Edge features are represented by constructing edge feature vectors. This paper suggests dividing the edge feature vector into two parts: one for edge direction and the other for edge information.

The part storing the direction information can be represented by a two-dimensional vector: the direction vector from node i to the adjacent edge of node j is marked as $[1, 0]$ on the adjacent edge of node i, and $[0, 1]$ on the adjacent edge of node j. We can get edge feature information by calculating the similarity coefficient of between two nodes, the in-degree, out-degree, the degree distribution of two nodes, or the Mahalanobis distance of two nodes in vector space.

3.2.2 Improvement of attention mechanism

The attention mechanism allows the central node to selectively aggregate information from neighbor nodes, enhancing local graph convolution efficiency. However, current graph attention mechanisms neglects edge features. Incorporating edge features into the attention mechanism enhances information transmission efficiency, improves network node learning, and strengthens structural information learning.

The attention mechanism of introducing edge features proposed in this paper is as follows:

$$b_{ij} = LeakyReLU\left(\mathbf{a}^T \left[\mathbf{W}\mathbf{h}_i || \mathbf{W}\mathbf{h}_j || \mathbf{W}^e \mathbf{x}_{ij}^e\right]\right) \quad (5)$$

The proposed enhancement involves treating the adjacent edge feature vector between two nodes as equally important as the node feature vector in the attention calculation. This ensures that the attention coefficient incorporates both node-node associations and adjacent edge features. This strengthens the selection of valuable neighbor information by the attention mechanism without polluting node information and impacting downstream tasks. Under this new mechanism, the weight calculation for information aggregation is defined as follows:

$$\alpha_{ij} = softmax_j\left(b_{ij}\right) = \frac{\exp\left(LeakyReLU\left(\mathbf{a}^T\left[\mathbf{Wh}_i||\mathbf{Wh}_j||\mathbf{W}^e\mathbf{x}^e_{ij}\right]\right)\right)}{\sum\limits_{k\in N_i} exp\left(LeakyReLU\left(\mathbf{a}^T\left[\mathbf{Wh}_i||\mathbf{Wh}_k||\mathbf{W}^e\mathbf{x}^e_{ik}\right]\right)\right)} \tag{6}$$

3.2.3 Model architecture

To apply GAT-Edge model for node classification, the process is as follows:

Feature processing and construction: Align the graph data's adjacency matrix with the node vector matrix and normalize it. Construct the edge eigenvector matrix, determine edge direction based on the adjacency matrix, and fill in direction vectors. Create a mask matrix to limit weighted aggregation to first-order neighbors.

Calculate multi-head attention: Using the processed node feature matrix, adjacency matrix, adjacent edge feature matrix, and mask matrix as inputs to the convolution layer, compute multi-head attention for each adjacent edge. Calculate final weights using multi-head attention.

Update central node feature vector: Weight and aggregate first-order neighbors based on calculated weights and the mask matrix. The central node's feature vector is updated, serving as the output for this hidden layer and input for the next one.

Output node prediction: Apply a fully connected neural network to the last hidden layer's output vector and use softmax to make the final node prediction.

3.2.4 The computational complexity of the model

In the process of node mapping, the main function of $\mathbf{Wh_j}$ is to map the vector of dimension d to the space of dimension $d^{'}$, so the computational complexity is $O(d \times d^{'})$. Because each node needs to be mapped with features, the actual computational complexity caused by the mapping process of nodes is $O(n \times d \times d^{'})$.

In the mapping process of edge features, the main function of $\mathbf{W^e x^e_{ij}}$ is to map a dimensional vector c to a dimensional vector space $c^{'}$, so its computational complexity is $O(c \times c^{'})$. Similarly, each adjacent edge needs to be mapped, so the actual computational complexity is $O(m \times c \times c^{'})$.

In the calculation of the attention mechanism, the main function of $a^T(\bullet)$ is to map the vector of a dimension $2 \times d^{'}$ to a real number. Unlike GAT, because it needs to map edge eigenvectors to real numbers, its actual computational complexity is $O\left(d^{'} + c^{'}\right)$. In the actual calculation process, the attention coefficient needs to be calculated for each adjacent edge, so the computational complexity of this process is $O(m \times \left(d^{'} + c^{'}\right))$.

Subsequent calculations are mainly weighted summation operations, so the computational complexity of the model will not be affected in essence. To sum up, the computational complexity of the GAT-Edge model is $O\left(n \times d \times d^{'}\right) + O\left(m \times c \times c^{'}\right) + O(m \times \left(d^{'} + c^{'}\right))$.

Compared with the GCN with $O\left(n^3\right)$, GAT-Edge significantly reduces the computational complexity, thus supporting large-scale computing. However, compared with the GAT model, the GAT-Edge model has increased the computational complexity a little bit due to the introduction of edge features in attention calculation.

4 Experiments

4.1 Data Sets

The experiments were conducted on three real-world academic paper citation networks: Cora, Citeseer, and PubMed. The Cora dataset comprises 2708 nodes and 5429 adjacent edges, with each node corresponding to a machine learning academic paper. These papers are divided into seven categories. Each paper is represented by a 1433-dimensional word vector, with 0 and 1 values indicating word presence or absence, after stemming, stop-word removal, and filtering low-frequency words.

The Citeseer and PubMed datasets underwent a similar preprocessing process. Citeseer has 3327 academic paper nodes, 4732 adjacent edges, and 3703-dimensional feature word vectors. PubMed consists of 19717 academic paper nodes, 44338 adjacent edges, and 500-dimensional feature word vectors. Papers in Citeseer are divided into six categories, while PubMed has three categories.

Due to limited computing resources, a random

	Cora	Citeseer	Sub PubMed
Number of nodes	2708	3327	4000
Number of edges	5429	4932	12415
Number of features on each node	1433	3703	500
Average degree of nodes	2.00	1.48	3.10
Network density	0.0014	0.0008	0.0016
Number of features on each edge	3	3	3
Categories of nodes	7	6	3
Number of nodes in training set	1625	1996	2400
Number of nodes in validation set	542	666	800
Number of nodes in test set	541	665	800

Table 2: Basic statistical information of the three datasets.

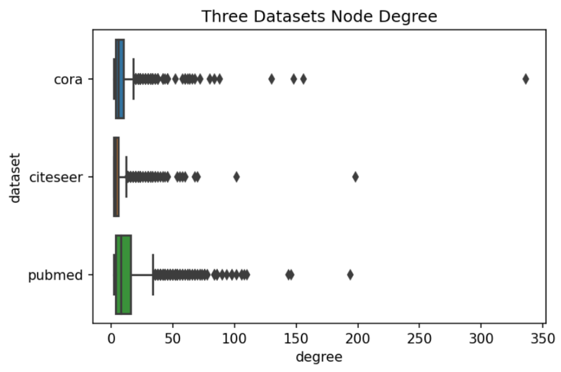

Figure 1: Box plots of node degree

walk sampling method was used to create a subgraph of PubMed data with 4000 nodes as a substitute for the full PubMed dataset during model training. The summary statistics of these three graph data are shown in Table 2.

The three citation network datasets are relatively sparse, with the largest Sub PubMed dataset having an average node degree of only 3.1, meaning each node has an average of three first-order neighbors. The smallest Citeseer dataset has an average node degree of less than 1.5, indicating that its topological structure relies less on graph information and more on node eigenvectors. In terms of network density, all three datasets are relatively low, which is very common in real graph data.

Figure 1 shows the node degrees which ranges from 0 to 10, decreasing rapidly with higher degrees. Sub-PubMed has a relatively flatter distribution with more high-degree nodes compared to Cora and Citeseer. There are many outliers, with the largest node having a degree of about 200, even 336 in Cora.

The edge features for these three academic paper citation networks are constructed as follows. For position vectors, it fills with either [0,1] or [1,0] based on the direction of the original adjacent edge. Edge feature vectors are obtained by calculating the

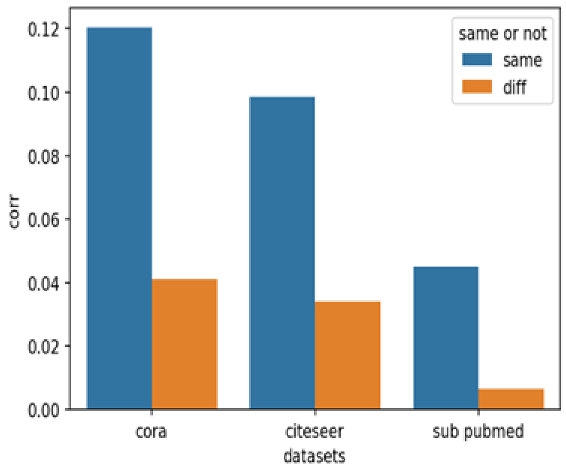

Figure 2: The correlation coefficient of the node features

correlation coefficient of feature vectors between two nodes, creating a three-dimensional vectors to represent edge information.

Since it is necessary to use the correlation coefficient of feature vectors between nodes to construct edge features, it's intuitive to expect that if node features effectively represent node categories, the correlation coefficient of feature vectors for nodes with the same category label should be higher, while nodes with different category labels should have a lower correlation coefficient. We calculate the correlation coefficients between nodes with the same label and those with different labels in the three datasets. The results are as expected as shown in Figure 2.

4.2 Hyperparameter Setting and Running Environment

The neural network, while powerful, requires manually setting hyperparameters, making it a cumbersome process, especially for large-scale datasets. Key hyperparameters for the GAT-Edge model include batch size, hidden layer size, number of attention heads, learning rate, L2 regularization coefficient.

The learning rate is a crucial parameter, balancing convergence speed and overfitting. The right learning rate improves model performance. Neural network size depends on hidden layer neuron count and depth. Wider networks, meaning more neurons in the hidden layer and fewer layers, are often better at capturing information, while depth incurs more computational cost.

Neural networks, due to their strong learning capacity, are susceptible to overfitting. To combat

Model parameter	Value
Batch Size	1
Hidden layers	2
Number of neurons in hidden layer	8
Number of heads of multiple attention mechanisms	8
Activate function	LeakyReLU
Optimizer	Adam
Drop out	0.6
Learning rate	0.01
Coefficient of L2 loss	0.0001

Table 3: Specific super parameter settings.

this, regularization terms like L1 and L2 regularization are added to the loss function, ensuring the network doesn't learn noisy data.

$$L1: \quad J(w,b)=\frac{1}{m}\sum_{i=1}^{m}L\left(\widehat{y}^{(i)},y^{(i)}\right)+\frac{\lambda}{2m}\|w\|_1 \tag{7}$$

$$L2: \quad J(w,b)=\frac{1}{m}\sum_{i=1}^{m}L\left(\widehat{y}^{(i)},y^{(i)}\right)+\frac{\lambda}{2m}\|w\|_2^2 \tag{8}$$

L1 regularization pushes weights towards 0, whereas L2 regularization shrinks weights close to 0. L2 is preferred in neural networks as it encourages neurons to capture more features. It punishes high neuron weights, preventing overfitting. Given the edge feature addition, the original GAT's L2 weight of 0.0005 is reduced to 0.0001 for better accuracy.

Table 3 outlines the experiment's specific parameter settings:

The running environment of the experiment is Ubuntu 16.04 LTS, CUDA 11.2, a TITAN V 12GB GPU, Python 3.6.5, and TensorFlow-gpu version 1.14.0.

4.3 Comparison Results

This paper compares the GAT-Edge model's performance with GCN, GAT, MPNN, AGNN and SPLINE. GCN, GAT and MPNN have been introduced in Section 2. AGNN (Attention-based Graph Neural Network, (Thekumparampil et al., 2018)) introduces an attention mechanism in the propagation layer to differentiate the attention of neighboring nodes during the aggregation process of central node features. SPLINE(Fey et al., 2017) utilizes Continuous B-Spline kernels to make the computation time independent from the kernel size due to the local support property of the B-spline basis functions.

Method	Cora	Citeseer	Sub PubMed
GCN	82.38%(2.5%)	74.89%(1.3%)	86.11%(0.8%)
GAT	86%(0.6%)	74.73%(1.1%)	85.62%(1.1%)
MPNN	85.9%(1.4%)	73.46%(1.1%)	84.69%(0.4%)
AGNN	85.60%(0.9%)	73.88%(1.1%)	86.56%(0.4%)
SPLINE	86.76%(1.2%)	74.62%(1.7%)	86.68%(0.8%)
GAT-Edge	**87.31%(0.8%)**	**76.50%(1%)**	**87.05%(1.5%)**

Table 4: Summary of experimental results.

	Cora	Citeseer	Sub PubMed
GAT-Edge Vs GAT	3.88*	3.48*	2.33*
GAT-Edge Vs GCN	5.68*	2.96*	1.67
GAT-Edge Vs SPLINE	1.17	2.95*	0.68
GAT-Edge Vs MPNN	2.75*	6.42*	4.78*
GAT-Edge Vs Agnn	4.61*	5.44*	1.01

*significant at 5% level

Table 5: Independent sample t-test of experimental results.

Each dataset is randomly divided into training, validation, and test sets in a 3:1:1 ratio with 10 random seeds. The datasets are split 10 times, and each model, including the comparisons, is run 10 times. Because our datasets do not have imbalance problem, multi-class classification accuracy is used as the evaluation metric. It is summarized in Table 4.

The average performance of the GAT-Edge model on the three datasets surpasses the comparison models based on the results of ten experiments. This indicates the effectiveness of mining opposing information in node classification using GAT-Edge.

To mitigate potential experiment bias, this paper considers conducting an independent sample t-test to assess whether the GAT-Edge's performance is significantly better than the other models. The null hypothese for the hypothesis test is as follows:

H_0: The accuracy of GAT-Edge model on the data set is not higher than that of the other model;

The results of the hypothesis test are shown in Table 5:

It can be seen from Table 5 that in most cases (11 out of 15), the null hypotheses are rejected at 5% significance level, so we believe that GAT-Edge significantly improved the prediction accuracy on the three data sets.

5 Conclusion

This paper introduces the GAT-Edge model, which addresses a significant gap in graph neural networks (GNN) research by incorporating edge features and direction to enhance graph data representation. While traditional GNNs focus on aggregating node information for downstream tasks, GAT-Edge goes beyond by considering edge attributes.

GAT-Edge leverages an attention mechanism to guide information aggregation, leading to superior performance in node classification experiments on Cora, Citeseer, and Sub PubMed datasets when compared to existing GNN methods.

6 Limitations

The GAT-Edge has some limitations: it may require substantial memory space and computational resources when handling a large number of high-dimensional edge features. Additionally, the model's reliance on correlation coefficients for edge feature construction can lead to instability with limited data.

Moreover, the model doesn't account for essential topological graph characteristics like ternary closure and shortest path, which could enhance its capacity to process graph data.

Finally, the paper could explore more efficient and interpretable methods for introducing edge features, offering promising avenues for further research.

Acknowledgement

This work is supported by National Natural Science Foundation of China (No. 72171229), and the MOE Project of Key Research Institute of Humanities and Social Sciences (No. 22JJD110001).

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. Neural machine translation by jointly learning to align and translate. *Preprint*, arXiv:1409.0473.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. 2017. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. *CoRR*, abs/1711.08920.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR.

Liyu Gong and Qiang Cheng. 2019. Exploiting edge features for graph neural networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9203–9211.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial networks. *Preprint*, arXiv:1406.2661.

Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. 2011. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *Preprint*, arXiv:1609.02907.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph classification using structural attention. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 1666–1674, New York, NY, USA. Association for Computing Machinery.

Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. 2018. Adaptive graph convolutional neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Duong Chi Thang, Hoang Thanh Dat, Nguyen Thanh Tam, Jun Jo, Nguyen Quoc Viet Hung, and Karl Aberer. 2022. Nature vs. nurture: Feature vs. structure for graph neural networks. *Pattern Recognition Letters*, 159:46–53.

Kiran K. Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. 2018. Attention-based graph neural network for semi-supervised learning. *Preprint*, arXiv:1803.03735.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. *Preprint*, arXiv:1710.10903.

FeiFei Wei, Mingzhu Ping, and KuiZhi Mei. 2022. Structure-based graph convolutional networks with frequency filter. *Pattern Recognition Letters*, 164:161–165.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057, Lille, France. PMLR.