Global and Stochastic Analysis
Vol. 12 No. 1 (January, 2025)

Received: 18th July 2024 **Revised: 25nd August 2024** **Accepted: 10th September 2024**

MESSAGE PASSING GRAPH NEURAL NETWORKS: A STUDY

SIMRAN, AMITESH PURI, AND S.AKANSHA

Abstract. Graph Neural Networks (GNNs) have emerged as powerful tools for learning representations of graph structured data, finding applications in diverse fields such as social network analysis, recommendation systems, and molecular chemistry. In this master's thesis, we present a thorough examination of state-of-the- art message-passing neural networks, including but not limited to Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), GraphSAGE, and Approximate Personalized Propagation of Neural Predictions (APPNP). Our study begins with a detailed exploration of the foundational concepts and theoretical underpinnings of message-passing neural networks. We delve into the fundamental mechanisms of information propagation across nodes within a graph, elucidating the various aggregation and diffusion strategies employed by different architectures. Subsequently, a comprehensive review of existing GNN models is undertaken, providing a comparative analysis of their strengths, weaknesses, and unique characteristics. This work contributes a concise yet comprehensive overview, aiding researchers and practitioners in understanding the evolving landscape of graph representation learning.

1. Introduction

In recent years, the explosion of data in various domains has led to an increased interest in harnessing the power of graph structures for modeling complex relationships [10, 7, 2, 4, 5]. Graphs, which consist of nodes and edges representing entities and their connections, respectively, have emerged as a fundamental data representation in fields such as social networks [7], recommendation systems, biology, chemistry, and more. As the diversity and complexity of graph-structured data grow, so does the demand for advanced tools to analyze and understand these intricate relationships.

This surge in interest has sparked the development of a remarkable class of machine learning models known as Graph Neural Networks (GNNs) [22, 24, 19]. GNNs are a novel approach to learning representations from graph-structured data, enabling us to capture both local and global information of nodes in a unified manner [21, 11]. In essence, GNNs extend the neural network architecture to accommodate graph data, where nodes represent entities and edges denote relationships. This extension opens the door to a multitude of applications, ranging from node classification [20, 8, 15] and link prediction to graph-level tasks like

Key words and phrases. Graph Neural Networks (GNNs), Graph convolutional network, Neural networks, Semi supervised learning,Message passing neural networks.

community detection [1, 9] and molecular property prediction [17, 16]. GNNs leverage the underlying graph structure to enable information propagation and aggregation, enabling them to capture intricate patterns that traditional machine learning models struggle to discern.

This article aims to provide a comprehensive panorama of the most popular form of GNNs, which is message passing neural networks. We present a thorough examination of state-of-the-art message-passing neural networks, including but not limited to Graph Convolutional Networks (GCN) [6], Graph Attention Networks (GAT) [13], GraphSAGE [5], and Approximate Personalized Propagation of Neural Predictions (APPNP) [3]. Our study begins with a detailed exploration of the foundational concepts and theoretical underpinnings of message-passing neural networks. We delve into the fundamental mechanisms of information propagation across nodes within a graph, elucidating the various aggregation and diffusion strategies employed by different architectures. Subsequently, a comprehensive review of existing GNN models is undertaken, providing a comparative analysis of their strengths, weaknesses, and unique characteristics. This work contributes a concise yet comprehensive overview, aiding researchers and practitioners in understanding the evolving landscape of graph representation learning.

2. Background

Semi-supervised learning Semi-supervised learning (SSL) is a machine learning technique that combines elements of both supervised and unsupervised learning. It operates on a dataset that is partially labeled. The primary objective of SSL is to address the limitations of purely supervised and unsupervisedmethods. Supervised learning typically requires a large amount of labeled data to accurately classify test data, which can be both costly and time-consuming to obtain. In contrast, unsupervised learning does not require labeled data and groups data points based on similarity, but it often fails to accurately classify unknown data.

SSL offers a solution to these challenges by leveraging a small amount of labeled data alongside a larger pool of unlabeled data. This approach enables the model to learn from the labeled examples and generalize to label the unlabeled data effectively. SSL constructs a model using a limited set of labeled patterns as training data, while the remaining patterns are treated as test data. In this article, our focus in on semi-supervised classification in graph neural networks.

2.0.1. *Semi-Supervised Classification.* Semi-Supervised Classification (SSC) is an approach similar to supervised learning but requires less training data to classify a large amount of test data. By utilizing SSC, the dependency on extensive training data is reduced. In the research community, a significant amount of unlabeled data is readily available, whereas labeled data is scarce due to the cost and time associated with generating it [12]. In [14], the authors proposed an approximation solution for labeling test patterns using a selective incremental transductive nearest neighbor classifier (SI-TNNC). They compared their results across five diverse datasets and five different algorithms, demonstrating that SI-TNNC achieved higher accuracy than standard algorithms like ID3 and 3NN in three out of five cases.

2.1. Message Passing Graph Neural Networks (MPGNN). A GNN is a neural network architecture designed to operate on graph-structured data. The core idea of a GNN is to iteratively aggregate information from neighboring nodes and update the node features through multiple layers. Consider a graph denoted as $G(V, E)$, where V represents the set of nodes and E is the set of edges (or links), with $E \subseteq V \times V$. In the context of Graph Neural Networks (GNNs), the primary objective is to learn effective representations for nodes, links, and even entire graphs. This is achieved through a fundamental process called message-passing, as defined by Gilmer et al. [4] and elaborated by Zhang et al. [23].

In matrix form, a graph is represented by an adjacency matrix A, which is an $n \times n$ matrix where A_{uv} denotes the connection between nodes u and v. If there is an edge between nodes u and v, then $A_{uv} = 1$; otherwise, $A_{uv} = 0$. Let D be a diagonal matrix with $D_{v,v} = \sum_u A_{vu}$, where $D_{v,v}$ represents the degree of the v-th vertex. A_{vu} represents the adjacency matrix element corresponding to the connection between vertices v and u, with each entry corresponding to the sum of the rows in the adjacency matrix, thereby ensuring a normalized representation.

Once we have the adjacency matrix of a graph, we proceed to the message aggregation phase. In this step, each node aggregates information from its neighboring nodes. The process is described by the following equation:

$$m_k^v = \sum_{u \in N(v)} M_k(h_v^{(k-1)}, h_u^{(k-1)}, e_{uv})$$

Here, $N(v)$ denotes the neighborhood of node v, $h_v^{(k-1)}$ represents the representation of node v at the $(k-1)$-th layer, $h_u^{(k-1)}$ represents the representation of node u at the $(k-1)$-th layer, and e_{uv} is the edge connection from node u to v. M_k is the message function that takes these inputs and generates the message to be passed from each neighboring node u to node v. m_k^v represents the aggregated message passed from all neighboring nodes u to v. After the aggregation process, the node features, including those from neighboring nodes, are combined through a weighted sum or a learned aggregation function:

$$a_k^v = \text{AGGREGATE}(m_k^v \cup h_v^{(k-1)})$$

Here, AGGREGATE combines the messages and previous features, typically using mean or sum. The nodes then update their hidden representations based on the aggregated message from their neighbors and their own feature vector. This update process is described by the following equation:

$$h_k^v = u_k(h_v^{(k-1)}, a_k^v)$$

In this equation, a_k^v is the aggregated message, and u_k is the update function that concatenates the aggregated message with the current node representation. The final node representations h_k^v after k layers can be used for downstream tasks:

$$h_v^{(k)} = \text{COMBINE}(h_v^{(k-1)}, \text{AGGREGATE}(h_u^{(k-1)} | u \in N(v)))$$

In this equation, $h_v^{(k-1)}$ represents the representation of node v at the $(k-1)$-th layer, and $N(v)$ signifies the set of neighbors of node v. This final representation

$h_v^{(k)}$ captures the combined information from both the node's own features and its neighborhood, which can then be utilized for various tasks such as node classification or link prediction.

3. Graph Convolutional Network (GCN)

A Graph Convolutional Network (GCN) is a type of neural network designed to operate on graph-structured data. It uses convolutional operations to process and analyze the relationships between nodes in a graph. Consider the graph $G(V, A)$, where $A \in \mathbb{R}^{n \times n}$ represents an adjacency matrix characterized by its symmetry, indicating the presence of edges between nodes. In GCNs, the aggregation process involves computing the average of the features of neighboring nodes. This is facilitated by leveraging the Degree Matrix D:

$$D^{-1}A = \text{AVERAGE}$$

where D^{-1} is the inverse of the degree matrix, and A is the adjacency matrix. In a GCN, each node v is associated with an initial feature vector represented by x_v. For the k-th graph convolution layer, the input node features are represented by $H^{(k-1)}$, and the output node features are represented by $H^{(k)}$ [18]: $H^{(0)} = x$. This indicates the first input layer of the GCN.

The message passing phase involves aggregating information from neighboring nodes. The aggregate and combine functions are represented by the following equation:

$$H_v^{(k)} = \sum_{u=1}^{n} a_{vu} \left(\frac{1}{\sqrt{d_v}} \frac{1}{\sqrt{d_u}} \right) h_u^{(k-1)} w_k + \frac{1}{d_v} h_v^{(k-1)} w_k$$

Here, the AGGREGATE function calculates the average value of the neighboring node representations. Each neighbor's contribution is weighted based on the weight of the edge between the node and its neighbor. This weight is determined by dividing the weight of the edge by the degrees of the two nodes involved, represented by w_k. The COMBINE function takes the aggregated messages from the neighbors and adds them to the node's own representation, normalized by its own degree [6]. This equation can be normalized using a simple matrix operation:

$$S = D^{-\frac{1}{2}} \tilde{A} D^{-\frac{1}{2}}$$

where $\tilde{A} = A + I$ is the adjacency matrix with self-loops, and I is the identity matrix. The final feature representation $H^{(k)}$ is then generated by applying a nonlinear activation function, commonly ReLU, pointwise:

$$H^{(k)} = \text{ReLU}(H_v^{(k)} W_k)$$

The pointwise nonlinear transformation is applied to the features in the k-th layer, and these transformed features are then propagated to the $(k+1)$-th layer [18].

4. Graph Sampling and Aggregation (GrahSAGE)

operates by sampling and aggregating data from each node's neighborhood in a graph. The primary idea behind GraphSAGE is to create node embeddings by gathering information from neighboring nodes. Each node v is associated with features x_v, and the goal is to learn node embeddings for each node. The aggregation process involves the following steps: For each node v, a sample of K neighbors is taken from its neighborhood, denoted as $N(v)$. The mean of the features of the neighboring nodes is computed. This mean aggregator is similar to the Graph Convolutional Network (GCN) framework [5]. The overall process can be described using the following equation:

$$h_v^{(k)} = \sigma\left(W \cdot \text{mean}\left(\left\{h_u^{(k-1)} \mid u \in N(v)\right\} \cup \left\{h_v^{(k-1)}\right\}\right)\right)$$

where $h_v^{(k)}$ represents the embedding of node v at layer k, W is a weight matrix, MEAN denotes the mean aggregation function, and σ is an activation function.

An alternative, more complex aggregator examined in GraphSAGE is based on the LSTM (Long Short-Term Memory) architecture, which has a larger expressive capability compared to the mean aggregator. However, LSTMs are not naturally symmetric since they process inputs sequentially. To adapt LSTMs for use with an unordered set, they are applied to a random permutation of the node's neighbors:

$$h_v = \text{LSTMAgg}(v, P(N(v)))$$

where h_v represents the aggregated representation for node v, LSTMAgg is the LSTM aggregation function, and $P(N(v))$ is a random permutation of the neighbors of v. The aggregated representations are passed through a fully connected layer, activated, and max-pooled to aggregate information across neighbors:

$$k = \max\left(\sigma\left(W_{\text{pool}} h_u^{(k)} - b\right), \forall u \in N(v)\right)$$

where k is the final aggregated representation, W_{pool} is a weight matrix for the pooling operation, $h_u^{(k)}$ is the representation of neighbor u at layer k, and b is a bias term. These steps outline the key processes in GraphSAGE for generating node embeddings by sampling and aggregating information from neighboring nodes.

5. Graph Attention Network (GAT)

Graph Attention Networks (GATs) enhance node embeddings by assigning different importance weights to neighboring nodes through attention mechanisms, leading to more informative and effective representations for downstream tasks. The process involves several key steps: We begin by setting up the node representations, denoted as h_0^u. These initial representations can be initialized randomly or based on predefined features. Let the initial hidden layer representation for every node be defined with dimension F:

$$h = \left(\vec{h}_1, \vec{h}_2, ..., \vec{h}_N\right), \quad \vec{h}_v \in \mathbb{R}^F$$

where N is the number of nodes. The output of this initial hidden layer is obtained by passing it through the transformation layer:

$$h_1^u = \sigma\left(\sum_{v \in N(u)} W\vec{h}_v\right)$$

where σ is a non-linear activation function. This transformation produces a new set of node features:

$$h_1 = \left(\vec{h}_1^1, \vec{h}_2^1, \vec{h}_3^1, ..., \vec{h}_N^1\right), \quad \vec{h}_u^1 \in \mathbb{R}^{F_1}$$

where F_1 can be greater or smaller than F. To determine the attention coefficient, which quantifies how much each neighboring node contributes to the current node, we use a self-attention mechanism. For nodes u and v, where u is the current node and v is a neighboring node, the attention coefficient is defined as:

$$e_{uv} = a\left(W\vec{h}_u, W\vec{h}_v\right)$$

Here, a is a shared attention mechanism function, W is a weight matrix $W \in \mathbb{R}^{F_1 \times F}$, and e_{uv} indicates the relationship strength between nodes u and v. To normalize the attention coefficients, we use the softmax function:

$$\alpha_{uv} = \text{softmax}_v(e_{uv}) = \frac{e_{uv}}{\sum_{k \in N(u)} \exp(e_{uk})}$$

This can be further elaborated as:

$$\alpha_{uv} = \frac{\exp\left(\text{LeakyReLU}\left(a^T\left[W\vec{h}_u \| W\vec{h}_v\right]\right)\right)}{\sum_{k \in N(u)} \exp\left(\text{LeakyReLU}\left(a^T\left[W\vec{h}_u \| W\vec{h}_k\right]\right)\right)}$$

where α_{uv} is the attention coefficient, LeakyReLU is the Leaky ReLU activation function, $\|$ denotes concatenation, and a is a learnable parameter. Using the attention coefficients, the final node embedding for node u is updated as follows:

$$\vec{h}_u^1 = \sigma\left(\sum_{v \in N(u)} \alpha_{uv} W\vec{h}_v^1\right)$$

In summary, Graph Attention Networks (GATs) enhance node embeddings by assigning different importance weights to neighboring nodes through attention mechanisms. This leads to more informative and effective representations for downstream tasks.

6. Approximate Personalized Propagation Of Neural Predictions (APPNP)

APPNP leverages personalized propagation and PageRank on graph-structured data to enhance node embeddings and improve predictions. We start with initializing the node features $h_0^v = x_v$, where x_v denotes the initial feature vector of node

v. Initially, we compute PageRank through a random walk on the graph, where nodes closer to the target node have higher probabilities:

$$\pi_{pr} = A_{rw}\pi_{pr}$$

Here, $A_{rw} = AD^{-1}$ and D^{-1} is the inverse of the degree matrix. Introducing a teleport vector i_x, personalized PageRank refines node importance based on proximity to i_x:

$$\pi_{ppr}(i_x) = (1-\alpha)\tilde{A}\pi_{ppr}(i_x) + \alpha i_x$$

Resulting in:

$$\pi_{ppr} = \alpha(I_n - (1-\alpha)\tilde{A})^{-1}$$

where $\tilde{A} = A + I$ and I is the identity matrix. Nodes are initially predicted using a neural network f_θ with parameters θ, yielding predictions H:

$$H_{i,:} = f_\theta(X_{i,:})$$

APPNP uses iterative message passing steps to refine predictions across the graph:

$$Z^{(0)} = H$$
$$Z^{(k)} = (1-\alpha)\tilde{A}Z^{(k-1)} + \alpha H$$

where $\tilde{A}$ includes the self-loops, ensuring propagation consistency without constructing an $n \times n$ matrix directly. The final node predictions $Z^{(K)}$ are obtained by applying a softmax function to the last iteration of message passing:

$$Z^{(K)} = \text{softmax}\left((1-\alpha)\tilde{A}^K Z^{(K-1)} + \alpha H\right)$$

This approach efficiently integrates node features and graph structure, enhancing prediction accuracy while maintaining computational efficiency.

7. Discussion

In summary, Graph Convolutional Networks (GCNs) leverage convolutional operations on graph structures, allowing nodes to aggregate and propagate information through their neighborhoods, akin to traditional convolutional neural networks in image processing. They excel in tasks where local neighborhood information is crucial, such as node classification and link prediction. GraphSAGE extends the idea of neighborhood aggregation by sampling and aggregating information from a node's neighbors, enabling scalability to large graphs while maintaining representation quality. This method is particularly useful in scenarios where full graph information is impractical due to size constraints. Graph Attention Networks (GATs) introduce attention mechanisms to GNNs, allowing nodes to selectively attend to different neighbors based on learned importance weights. This adaptability enhances the model's ability to capture varying degrees of influence from neighboring nodes, improving performance in tasks requiring nuanced

relational reasoning. The Approximate Personalized Propagation of Neural Predictions (APPNP) integrates personalized PageRank with neural network predictions, facilitating effective information propagation across the graph. By iteratively refining node predictions through personalized propagation steps, APPNP achieves state-of-the-art results in tasks demanding personalized influence modeling and graph-wide prediction aggregation.

8. Conclusion

In this article, we explored four prominent types of Message Passing Graph Neural Networks (GNNs): Graph Convolutional Networks (GCNs), GraphSAGE, Graph Attention Networks (GATs), and the Approximate Personalized Propagation of Neural Predictions (APPNP). Each of these models represents a significant advancement in the field of graph-based learning, offering unique approaches to effectively capture and utilize relational information in graph-structured data. In conclusion, Message Passing Graph Neural Networks represent a powerful paradigm for learning from graph-structured data, offering versatile tools to tackle a wide range of tasks including node classification, link prediction, and graph-level tasks. As research continues to evolve, further advancements in GNN architectures promise to unlock new capabilities in understanding and analyzing complex relational data.

Funding Information

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Competing Interests

The authors have no relevant financial or non-financial interests to disclose.

References

References

1. Zhengdao Chen, Xiang Li, and Joan Bruna, *Supervised community detection with line graph neural networks*, arXiv preprint arXiv:1705.08415 (2017).
2. Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, *Convolutional neural networks on graphs with fast localized spectral filtering*, Advances in neural information processing systems **29** (2016).
3. Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann, *Predict then propagate: Graph neural networks meet personalized pagerank*, International Conference on Learning Representations (ICLR), 2019.
4. Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl, *Neural message passing for quantum chemistry*, International conference on machine learning, PMLR, 2017, pp. 1263–1272.
5. Will Hamilton, Zhitao Ying, and Jure Leskovec, *Inductive representation learning on large graphs*, Advances in neural information processing systems **30** (2017).
6. Thomas N Kipf and Max Welling, *Semi-supervised classification with graph convolutional networks*, arXiv preprint arXiv:1609.02907 (2016).
7. Jure Leskovec and Julian Mcauley, *Learning to discover social circles in ego networks*, Advances in neural information processing systems **25** (2012).

8. Hogun Park and Jennifer Neville, *Exploiting interaction links for node classification with deep graph neural networks.*, IJCAI, vol. 2019, 2019, pp. 3223–3230.
9. Chenyang Qiu, Zhaoci Huang, Wenzhe Xu, and Huijia Li, *Vgaer: graph neural network reconstruction based community detection*, arXiv preprint arXiv:2201.04066 (2022).
10. Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F Samatova, *Anomaly detection in dynamic networks: a survey*, Wiley Interdisciplinary Reviews: Computational Statistics **7** (2015), no. 3, 223–247.
11. Ryoma Sato, *A survey on the expressive power of graph neural networks*, arXiv preprint arXiv:2003.04078 (2020).
12. Philippe Thomas, *Review of" semi-supervised learning" by o. chapelle, b. schölkopf, and a. zien, eds. london, uk, mit press, 2006*, IEEE Transactions on Neural Networks **20** (2009), no. 3, 542–542.
13. Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio, *Graph attention networks*, arXiv preprint arXiv:1710.10903 (2017).
14. P Viswanath, K Rajesh, C Lavanya, and YCA Padmanabha Reddy, *A selective incremental approach for transductive nearest neighbor classification*, 2011 IEEE Recent Advances in Intelligent Computational Systems, IEEE, 2011, pp. 221–226.
15. Yangkun Wang, Jiarui Jin, Weinan Zhang, Yong Yu, Zheng Zhang, and David Wipf, *Bag of tricks for node classification with graph neural networks*, arXiv preprint arXiv:2103.13355 (2021).
16. Zhengyang Wang, Meng Liu, Youzhi Luo, Zhao Xu, Yaochen Xie, Limei Wang, Lei Cai, Qi Qi, Zhuoning Yuan, Tianbao Yang, et al., *Advanced graph and sequence neural networks for molecular property prediction and drug discovery*, Bioinformatics **38** (2022), no. 9, 2579–2586.
17. Oliver Wieder, Stefan Kohlbacher, Mélaine Kuenemann, Arthur Garon, Pierre Ducrot, Thomas Seidel, and Thierry Langer, *A compact review of molecular property prediction with graph neural networks*, Drug Discovery Today: Technologies **37** (2020), 1–12.
18. Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger, *Simplifying graph convolutional networks*, International conference on machine learning, PMLR, 2019, pp. 6861–6871.
19. Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip, *A comprehensive survey on graph neural networks*, IEEE transactions on neural networks and learning systems **32** (2020), no. 1, 4–24.
20. Shunxin Xiao, Shiping Wang, Yuanfei Dai, and Wenzhong Guo, *Graph neural networks in node classification: survey and evaluation*, Machine Vision and Applications **33** (2022), 1–19.
21. Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka, *How powerful are graph neural networks?*, arXiv preprint arXiv:1810.00826 (2018).
22. Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji, *Explainability in graph neural networks: A taxonomic survey*, IEEE transactions on pattern analysis and machine intelligence **45** (2022), no. 5, 5782–5799.
23. Yongqi Zhang and Quanming Yao, *Knowledge graph reasoning with relational digraph*, Proceedings of the ACM Web Conference 2022, 2022, pp. 912–924.
24. Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun, *Graph neural networks: A review of methods and applications*, AI open **1** (2020), 57–81.

Department of Mathematics, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal-576104, India
Email address: simranmoily01@gmail.com

Department of Mathematics, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal-576104, India
Email address: amiteshpuri007@gmail.com

Department of Mathematics, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal-576104, India
Email address: akansha.agrawal@manipal.edu