Diffusion-based graph representation learning algorithms: Graph Diffusion Network (GDN), Graph Convolutional Networks (GCN) and Graph Diffusion System (GDS)

B.Malathi[1], Dr.S.Chelliah[2*]

[1]Research Scholar (Reg. no 1921072092017), [1,2] PG and Research Department of Mathematics, The M.D.T. Hindu College, Affiliated to ManonmaniamSundaranar University, Tirunelveli-627010, Tamilnadu, India.

[*]bsmalathi17512@gmail.com

Abstract:

Graph representation learning has emerged as a crucial task in the field of machine learning due to the widespread use of graphs in various domains such as social networks, biological networks, and citation networks. The primary goal of graph representation learning is to learn low-dimensional representations of graph nodes that can capture the underlying structure of the graph and be used for downstream tasks such as node classification, link prediction, and graph clustering.However, traditional graph representation learning methods rely on the assumption of local smoothness, which may not be sufficient for dealing with large, complex graphs where the local structure may not fully capture the global structure. To overcome this limitation, advanced methods such as the Graph Diffusion Network (GDN) have been proposed to capture both local and global information.GDN is designed to maintain both local and global consistency of the graph, and it does so by using a graph diffusion system to control the random walk of information flow and sense high-order local relationships in the graph. This approach allows GDN to capture both local and global information and learn intrinsic node representations in a progressive manner.One of the key advantages of GDN is its ability to self-refine on the graph structure, which enables it to learn the intrinsic node representations in a more efficient and effective manner compared to traditional methods that require the entire graph to be processed at once. Experiments on node classification tasks demonstrate that GDN outperforms traditional methods and achieves state-of-the-art results on several benchmark datasets.Overall, the success of GDN highlights the importance of capturing both local and global information in

complex graphs and has the potential to advance graph representation learning in various domains

Keywords:*Graph Diffusion Network (GDN), Graph Convolutional Networks (GCN) and GraphDiffusion System (GDS).*

1. Introduction

Diffusion-based graph representation learning algorithms are a class of methods that utilize the diffusion process on graphs to capture both local and global information. Here are brief descriptions of three popular diffusion-based graph representation learning algorithms: Graph Diffusion Network (GDN), Graph Convolutional Networks (GCN), and Graph Diffusion System (GDS).

1. Graph Diffusion Network (GDN): GDN is a graph neural network that is designed to learn low-dimensional representations of graph nodes by leveraging the graph diffusion process. It uses a diffusion system to propagate information through the graph, which allows it to capture both local and global information[1-5]. GDN maintains the local consistency of the graph by preserving the first-order proximity information and the global consistency of the graph by preserving the high-order proximity information. This makes it an effective method for graph representation learning.
2. Graph Convolutional Networks (GCN): GCN is a type of convolutional neural network (CNN) that operates directly on graphs. It uses a neighborhood aggregation strategy to propagate information through the graph, which allows it to capture both local and global information. GCN operates on the spectral domain of the graph and utilizes graph Laplacian matrices to perform convolution operations. This makes it an effective method for graph representation learning.
3. Graph Diffusion System (GDS): GDS is a diffusion-based graph representation learning algorithm that uses a graph diffusion process to propagate information through the graph. It is designed to capture both local and global information by using a diffusion system to control the random walk of information flow and sense high-order local relationships in the graph[6-10]. GDS utilizes a multi-scale approach to learn node representations, which allows it to capture information at different levels of granularity.

In summary, diffusion-based graph representation learning algorithms such as GDN, GCN, and GDS are effective methods for learning low-dimensional representations of graph nodes that can capture both local and global information[11-18]. These algorithms have been shown to achieve state-of-the-art results on various benchmark datasets and have the potential to advance graph representation learning in various domains

2. High-level algorithm for Graph Convolutional Networks (GCNs):

1. Define the graph structure and node features: a. Create an adjacency matrix A that represents the connections between nodes. b. Create a feature matrix X that represents the feature vectors associated with each node.
2. Initialize the weight matrices: a. Initialize a weight matrix *W1* that maps the input feature matrix *X* to a hidden feature matrix *H1*. b. Initialize a weight matrix *W2* that maps the hidden feature matrix *H1* to an output feature matrix *Y*.
3. Propagate the information between nodes: a. Compute the normalized adjacency matrix $\bar{A}$ by adding a self-loop to *A* and then normalizing each row. b. Compute the hidden feature matrix *H1* by applying the following equation: $H1 = ReLU(\bar{A}XW1)$ c. Compute the output feature matrix *Y* by applying the following equation: $Y = \bar{A}H1W2$
4. Train the GCN: a. Define a loss function that measures the error between the predicted output *Y* and the ground truth labels. b. Use backpropagation to update the weight matrices *W1* and *W2* to minimize the loss function.
5. Make predictions: a. Apply the trained GCN to new input data by propagating the information through the network using the updated weight matrices *W1* and *W2*. b. Use the output feature matrix *Y* to make predictions for the corresponding nodes.

Note: This is a simplified version of the GCN algorithm and there are variations and optimizations that can be applied based on the specific problem and data being analyzed

3. GDN algorithm

The Graph Diffusion Network (GDN) algorithm is a graph-based machine learning algorithm that is used for tasks such as node classification, link prediction, and graph clustering. It is based on the concept of diffusion, which involves spreading information across a graph by iteratively updating the node features.

Here's the algorithm:

1. Begin by initializing each node's feature vector with a one-hot encoding of its label. For example, if there are three classes *(A, B, C),* and node 1 belongs to class B, then its feature vector would be [0, 1, 0].

2. Define a diffusion matrix *W*, which encodes the strength of the connections between nodes. *W* can be defined in a variety of ways, such as using the adjacency matrix or a normalized Laplacian matrix.
3. Set a diffusion parameter alpha, which determines the rate of diffusion. Higher values of alpha will result in faster diffusion.
4. Define a number of diffusion steps *T*. Each diffusion step updates the feature vectors of all nodes.
5. Perform *T* diffusion steps by iterating the following formula:
6. $f^{(t+1)} = \alpha. W . f^{(t)} + (1 - \alpha) . f^{(0)}$
7. Here, $f^{(t)}$ represents the feature vector of all nodes at iteration *t*, and *f(0)* is the initial feature vector at iteration 0. The term $\alpha\ W . f^{(t)}$ represents the diffusion of information across the graph, while the term $(1 - \alpha) . f^{(0)}$ serves as a smoothing term that prevents the features from changing too drastically.
8. After *T* diffusion steps, use the resulting feature vectors to perform the desired task, such as node classification or link prediction.

The GDN algorithm is relatively simple to implement, yet it has been shown to perform well on a variety of graph-based machine learning tasks. The choice of diffusion matrix, diffusion parameter, and number of diffusion steps can all be adjusted to optimize performance for a particular task or dataset.

4. Graph diffusion system

The Graph Diffusion System (GDS) is a mathematical framework for analyzing and processing data that is represented as a graph. The goal of GDS is to obtain a smooth and low-dimensional representation of the graph that preserves the underlying structure of the data. Here is the algorithm and derivation for the Graph Diffusion System:

1. Construct a graph representation of the data, where the nodes represent data points, and the edges represent the relationships between them.
2. Compute the graph Laplacian matrix *L*, which is a matrix that encodes the structure of the graph. There are different ways to define the Laplacian matrix, but a common one is the normalized Laplacian, given by $L = I - D^{-1/2} A D^{-1/2}$, where *A* is the adjacency matrix of the graph, *D* is the diagonal degree matrix, and I is the identity matrix.

3. Define the diffusion process on the graph as $p(t) = e^{(-tL),}$ where t is a diffusion parameter that controls the smoothness of the resulting representation.
4. Compute the eigenvalues and eigenvectors of L, denoted by λ_i and v_i, respectively. The eigenvectors are sorted in increasing order of their corresponding eigenvalues.
5. Define the GDS representation of a data point x as a vector $y = (y_1, y_2, ..., y_k)$, where k is the desired dimensionality of the representation, and $y_i = \sum_j v_{ij} xj\ e^{(-t\lambda i)}$ is the i-th component of y, which weights the contribution of each data point x_j to the i-th eigenvector v_i, with a decay factor $e^{(-t\lambda_i).}$

The Graph Diffusion System is based on the diffusion process, which is a fundamental concept in physics that describes the spreading of particles or information through a medium. In the context of graphs, the diffusion process can be used to propagate information between data points, based on their similarity or proximity in the graph.

The diffusion process on a graph can be described by the heat equation:

$$\partial u/\partial t = -L\ u$$

where u(t) is the state of the system at time t, and L is the Laplacian matrix of the graph. The Laplacian matrix is a discrete version of the Laplace operator, which measures the curvature or smoothness of a function on a manifold.

The solution of the heat equation is given by the diffusion kernel:

$$p(t) = e\hat{}(-tL)$$

which represents the probability distribution of the diffusion process at time t. The diffusion kernel can be interpreted as a smoothing or blurring operator, which attenuates the high-frequency components of the data while preserving the low-frequency ones.

The GDS representation is based on the eigen decomposition of the Laplacian matrix, which decomposes the graph into a set of orthogonal eigenvectors and their corresponding eigenvalues. The eigenvectors represent the modes of variation or deformation of the graph, while the eigenvalues represent their magnitudes.

Graph diffusion systems (GDS) are a class of algorithms used for information diffusion and recommendation in graphs or networks. The basic idea behind GDS is to iteratively propagate information or recommendations throughout the network based on the graph structure and the content of the nodes.

Here is a high-level description of a typical GDS algorithm:

1. Initialize the node scores or weights. This can be done by setting all scores to 1, or by using some other method based on the node content or structure.
2. Define a diffusion function that describes how scores are propagated between nodes. This can be based on the edge weights, the node content, or some combination of the two. Common diffusion functions include heat kernel diffusion, PageRank, and personalized PageRank.
3. Run the diffusion function for a fixed number of iterations or until convergence. At each iteration, update the node scores based on the diffusion function and the current scores of neighboring nodes.
4. Normalize the node scores to ensure that they sum to 1, and use the resulting scores to make recommendations or predictions.

5. Single diffusion layer

A single diffusion layer algorithm is a computational method for solving partial differential equations that describe diffusion processes. Here's a basic algorithm for a single diffusion layer:

- Define the domain and boundary conditions of the problem.
- Discretize the domain into a grid of nodes.
- Initialize the solution at each node to an initial value.
- Calculate the time step, based on the stability criterion for the method being used.
- For each time step:

a. Calculate the values of the diffusion coefficient at each node, based on the solution at the previous time step.

b. Calculate the values of the Laplacian at each node, using the values of the solution at the previous time step.

c. Update the solution at each node, using the following equation

new_value = old_value + (diffusion_coefficient * time_step * Laplacian)

d. Apply any boundary conditions to the updated solution.

Repeat step 5 until the solution reaches a desired level of accuracy or convergence.

Note that this algorithm assumes a uniform diffusion coefficient and a single diffusion layer. In more complex scenarios, additional terms and boundary conditions may need to be included in the algorithm.

6. Total loss function of graph diffusion network

The Total Loss function for Graph Diffusion Network (GDN) is typically used for semi-supervised learning tasks on graphs. The goal is to learn a function that predicts node labels for all nodes in the graph, given a small set of labeled nodes.

The Total Loss function for GDN is defined as:

$$L\_total = L\_supervised + \alpha * L\_unsupervised$$

Where L_supervised is the supervised loss, which measures the error between the predicted labels and the ground truth labels for the labeled nodes. This can be any standard loss function, such as mean squared error or cross-entropy.

L_unsupervised is the unsupervised loss, which encourages the model to learn a smooth representation of the graph. This is typically done using a graph Laplacian regularization term, which penalizes changes in the model's predictions between neighboring nodes in the graph.

α is a hyperparameter that controls the trade-off between the supervised and unsupervised losses.The algorithm for calculating the Total Loss function for GDN is as follows:

1. Feed the input data into the GDN model and obtain the predicted node labels for all nodes in the graph.
2. Compute the supervised loss L, supervised using the predicted labels and the ground truth labels for the labeled nodes.
3. Compute the unsupervised loss L, unsupervised using the predicted labels and the graph Laplacian matrix.
4. Compute the Total Loss L, total as the sum of L, supervised and α times L, unsupervised.
5. Update the model parameters using backpropagation to minimize L, total.

Note that the Laplacian matrix can be computed using various methods, such as the standard graph Laplacian or the diffusion-based Laplacian. The choice of Laplacian matrix can affect the performance of the GDN model, and different methods may be appropriate for different graph structures.

7. Complexity analysis

Results of complexity analysis from efficiency graph diffusion network and graph convolutional network. Without specific details about the specific graph diffusion network and graph convolutional network being compared, it is difficult to provide a specific complexity analysis. However, I can provide some general insights on the efficiency graphs of these two network types.

Graph diffusion networks are a type of neural network that uses diffusion processes on a graph to propagate information across the graph. They are typically used for tasks such as semi-supervised learning and node classification. The efficiency of a graph diffusion network depends on the size and structure of the graph, as well as the parameters of the diffusion process used. In general, graph diffusion networks have a complexity of O(N^3), where N is the number of nodes in the graph.

Graph convolutional networks (GCNs) are another type of neural network that operate on graph data. GCNs use a convolutional-like operation on the graph to aggregate information from neighboring nodes. Like graph diffusion networks, the efficiency of GCNs depends on the size and structure of the graph, as well as the number of layers and the size of the feature vectors used. In general, the time complexity of a GCN is O(N^2) or O(N^3), depending on the implementation.

Overall, both graph diffusion networks and GCNs can be computationally expensive, particularly for large graphs. However, recent research has explored techniques for improving the efficiency of these networks, such as graph sparsification and parallelization.

8. Conclusion

Graph representation learning aims to learn low-dimensional representations of graph nodes for downstream tasks. Traditional methods rely on local smoothness, but advanced methods capture both local and global information. The Graph Diffusion Network (GDN) maintains local and global consistency by controlling information flow and sensing local relationships. GDN self-refines on graph structure and outperforms traditional methods on node classification tasks, highlighting the importance of capturing both local and global information

9. References

[1] Anuradha Gupta, and Manu Rohilla. "Fixed point theorems of (α, ψ) G-contractive mappings in quasi-partial b-metric-like spaces endowed with a graph." Asian-European Journal of Mathematics 14, no. 02 (2021): 2150014.

[2] Chicone, Carmen,Contraction,Ordinary Differential Equations with Applications (2nd ed.). New York: Springer, 121–135(2006).

[3] Coifman R.R and S. Lafon, Diffusion maps, Applied and Computational Harmonic Analysis , vol. 21(1)5–30 (2006).

[4] Coifman RR and Leeb WE, Earth Mover's distance and equivalent metrics for spaces with semigroups, Tech. Rep. YALEU/DCS/TR-1481(2013).

[5] Coifman RR and Maggioni M, Diffusion wavelets, Applied and Computational Harmonic Analysis , vol. 21(1)53–94(2006).

[6] Coifman RR, Lafon S, Lee et al AB, Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps, Proceedings of the National Academy of Sciences of the United States of America, vol. 102(21)7426–7431(2005).

[7] Daniel Fullmer, and A. Stephen Morse. "A distributed algorithm for computing a common fixed point of a finite family of paracontractions." IEEE Transactions on Automatic Control 63, no. 9 2833-2843(2018).

[8] Fick, A. Ueber Diffusion [On diffusion], Annalen der Physik und Chemie von J. C. Pogendorff(94)59-86(1855).

[9] Gabriel Connor Streich, Alexander J. Henshaw, James Brasington, Walter Bertoldi, and Gemma L. Harvey. "Let's get connected: A new graph theory-based approach and toolbox for understanding braided river morphodynamics." Wiley Interdisciplinary Reviews: Water 5, no. 5 1296(2018).

[10] Goldberg M J and Kim S, Some Remarks on Diffusion Distances, Journal of Applied Mathematics(2010) Article ID 464815(17)(2010).

[11] Goldberg M J and S. Kim, An efficient tree-based computation of a metric comparable to a natural diffusion distance, Applied and Computational Harmonic Analysis , vol. 33(2) 261–281 (2012).

[12] Graham T. A short account of experimental researches on the diffusion of gases through each other, and their separation by mechanical means, Quarterly Journal of Science, Literature, and Art, 2, 74–83(1829).

[13] Kamal Fallahi, Mujahid Abbas, and Ghasem Soleimani Rad. "Generalized c-distance on cone b-metric spaces endowed with a graph and fixed-point results." Applied General Topology 18, no. 2 (2017): 391-400.

[14] Kipf T.N, Welling.M ,Semi-supervised classification with graph convolutional networks, ICLR, (2017).

[15] Rokach L, Maimon O, Data mining and Knowledge Discovery Handbook, Springer, New Yk, NY, USA 321-352(2005).

[16] Stein EM, Topics in Harmonic Analysis Related to the Littlewood-Paley Theory, vol. 63 of Annals of Mathematics Studies, Princeton University Press, Princeton, NJ, USA, 3rd edition (1985).

[17] Sturm KT, Diffusion processes and heat kernels on metric spaces, Annals of Probability, vol. 26(1)1–55(1998).

[18] Vaibhav Mehandiratta, Mani Mehra, and Günter Leugering. "Existence and uniqueness results for a nonlinear Caputo fractional boundary value problem on a star graph." Journal of Mathematical Analysis and Applications 477, no. 2 (2019): 1243-1264.