Structured Data project: "Diffusion Improves Graph Learning"

Article written by :
Johannes Gasteiger
Stefan Weißenberger
Stephan Günnemann

Thibaut Valour
Godefroy DU CHALARD

Jury: Florence d'Alché-Buc and Jhony Giraldo

March 25th, 2023

1 Introduction

Graph Neural Networks (GNNs) have become a popular tool for performing machine learning tasks on graph-structured data, such as social networks, knowledge graphs, and molecular graphs [6]. However, the traditional approach of passing messages between one-hop neighbors in graph convolution has its limitations. For instance, in a social network, a user's influence may extend beyond their immediate neighbors to their neighbors' neighbors. Limiting graph convolution to one-hop neighbors would not capture such influence. This limitation highlights the need for a more powerful and flexible approach to graph convolution.

To address this limitation, researchers from the Technical University of Munich proposed in 2019 a new approach called Graph Diffusion Convolution (GDC) [2]. GDC leverages generalized graph diffusion to aggregate information from a larger neighborhood beyond one-hop neighbors, making it a more powerful and generalizable approach to graph convolution. By replacing traditional message passing with GDC, researchers have demonstrated consistent performance improvements across various supervised and unsupervised learning tasks on graphs. Moreover, GDC can be easily combined with other graph-based models or algorithms.

This report is divided into two main sections. Firstly, we will introduce the GDC method and its theoretical framework. Secondly, we will present the results of our implementation [1] of GDC, which was developed from scratch and tested on a node classification task.

[1] `https://github.com/thibautvalour/Graph-Diffusion-Convolution`

2 Analysis of the article

2.1 Framework of the study

Given a graph $G = (V, E)$, where V represents the nodes and E represents the edges, let $X \in R^{|V| \times F}$ be the input feature matrix, with F denoting the number of features for each node. The objective of Graph Neural Networks (GNNs), also known as Message Passing Layer Neural Networks or Graph Convolutional Networks, is to learn a function that maps the input features to an output embedding matrix $Z \in R^{|V| \times F'}$, where F' corresponds to the number of output features for each node. The graph convolution building block in GNNs can be defined as follows:

$$Z_{k+1} = \sigma(\tilde{A} Z_k W_k)$$

where $Z_0 = X$ is the input feature matrix, $\tilde{A}$ is a transformation of the adjacency matrix A of the graph, and W_k is a trainable matrix. σ is a non-linear activation function, often $ReLU$.

The matrix $\tilde{A}$ can be defined using the graph Laplacian matrix $L = D - A$, where D is the diagonal matrix of node degrees and A is the adjacency matrix of the graph. The normalized graph Laplacian matrix $L_{\text{norm}} = D^{-1/2} L D^{-1/2}$ can be used to obtain $\tilde{A}$ as follows:

$$\tilde{A} = L_{\text{norm}} + I_{|V|},$$

where $I_{|V|}$ is the $|V| \times |V|$ identity matrix. This ensures that each node is included in its own neighborhood.

The graph convolution operation can be stacked to form multiple layers. The operations that follow depend on the specific task we are working on. Node classification will be presented in more detail in the next section.

2.2 Definition of the Diffusion Matrix

The interest of this model result in the choice of the matrix $\tilde{A}$. The intuition behind GDC is to create a transition matrix that connects nodes which are close to each other. In a standard adjacency matrix, closeness means being connected by an edge. However, in some applications, edges might be arbitrary, and we may want to connect nodes that are second or third-degree neighbors. Gasteiger et al. proposed in [2] the following diffusion matrix for this purpose:

$$S = \sum_{k=0}^{\infty} \theta_k T^k \tag{1}$$

with the weighting coefficients θ_k , and the generalized transition matrix T. The challenge lies in determining the appropriate values for T and θ_k. Two conditions presented in the article ensure the convergence of this equation:

- The sum of the weighting coefficients must equal one: $\sum_{k=0}^{\infty} \theta_k = 1$, with $0 \leq \theta_k$.
- The eigenvalues of the transition matrix T must be bounded by lambda $\lambda \in [0, 1]$.

Indeed, since T is symmetric and real, it is diagonalizable as $T = PDP^{-1}$, where D is a diagonal matrix with eigenvalues of T. Using this, we have:

$$S = \sum_{k=0}^{\infty} \theta_k T^k = P \left(\sum_{k=0}^{\infty} \theta_k D^k \right) P^{-1} \tag{2}$$

Now, we focus on the series inside the parentheses:

$$\sum_{k=0}^{\infty} \theta_k D^k \tag{3}$$

This series represents a diagonal matrix, with diagonal elements converging absolutely due to the bounded eigenvalues of T ($\lambda \in [0, 1]$) and the sum of θ_k equaling 1. Therefore, the overall series converges, proving the convergence of the generalized graph diffusion.

2.3 Two main cases

Tow special cases presented in [2] meets the criteria presented above:

- **Personalized PageRank (PPR) [4]**: $\theta_k = \alpha(1-\alpha)^k$ with $\alpha \in [0, 1]$
- **Heat kernel [3]**: $\theta_k = e^{-t}\frac{t^k}{k!}$

There are multiple options for T, similarly to the various Laplacian matrices available, depending on the use case.

For the first case, we even have a simplification of the computing of S with the geometric sum:

$$\sum_{k=0}^{\infty} \alpha(1-\alpha)^k T^k = \alpha \sum_{k=0}^{\infty}((1-\alpha)T)^k = \alpha(I_{|V|} - (1-\alpha)T)^{-1} \tag{4}$$

2.4 Complete Method

The entire method involves more than just obtaining the diffusion matrix S. In fact, there are several steps to derive the final transition matrix. We will discuss each step in detail:

Computing T: The initial step involves calculating the generalized transition matrix T, which represents the direct relationship between nodes in the graph. There are multiple choices for T, such as $T_{rw} = AD^{-1}$, $T_{sym} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ and $\widetilde{T}sym = (wloopI_N + D)^{-\frac{1}{2}}(w_{loop}I_N + A)(w_{loop}I_N + D)^{-\frac{1}{2}}$, with A the adjacency matrix and D the degree matrix. Each of these options has its own properties and may be suitable for different use cases, depending on the characteristics of the graph being analyzed.

Computing S: Next, we compute the diffusion matrix S using the formulas discussed in the previous subsection.

Sparsification of the matrix: To enhance the efficiency of the method, sparsification is applied to the diffusion matrix S. In fact, after computing S, since the graph is connected, we will end up with a matrix with no zero elements. However, this is not ideal because nodes should not be all directly connected; we want to preserve close relationships through diffusion. To sparsify S, either set all elements below a threshold ϵ to zero or select only the k largest elements in each column of the matrix. In this case, k or ϵ become hyperparameters of the problem. By doing so, we obtain new edges that more effectively represent closeness compared to the edges in the initial graph.

Computing the final T_S: Lastly, we compute the final transition matrix T_S using the sparsified S, following the same approach as when obtaining T from A in the first step.

By following these steps, we obtain the final transition matrix that captures the diffusion process across the graph, accounting for the relationships between nodes. This matrix can now be used as a transition matrix in a Graph Convolutional Network. The following image from [2] sums up this process.

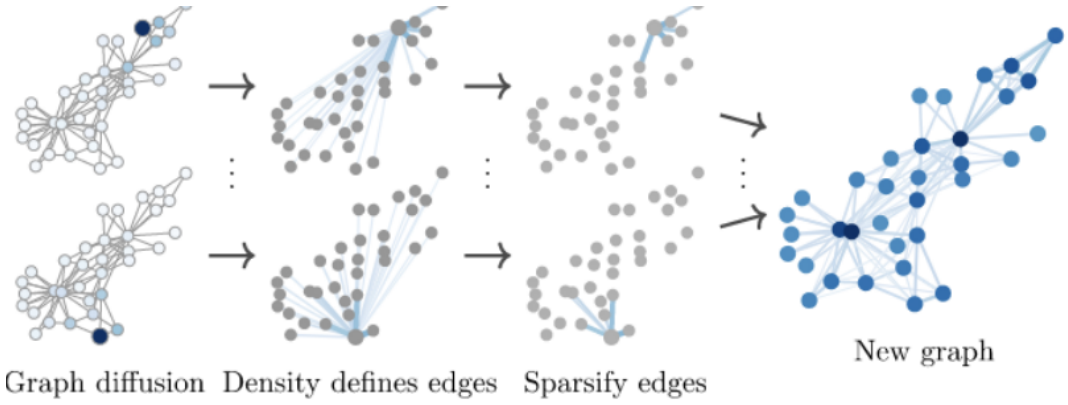

Figure 1: Schema of the GDN

2.5 Spectral analysis

While GDC is mainly based on spatial relationships, it can also be analyzed as a graph convolution in the graph spectral domain. In this domain, the graph can be interpreted as a polynomial filter. In fact, computing the diffusion matrix is equivalent to applying a low-pass filter on the graph, and computing the transition matrix is equivalent to applying a weak high-pass filter.

2.6 Complexity

When implementing the method from [2], computing the diffusion matrix S for a graph of size N requires calculating T^k for a few values of k (such as $k \in [1, 10]$) for the heat kernel. For PageRank, matrix inversion can be used instead of multiple matrix multiplications, as shown in equation (4). Both methods have a similar time complexity of around $O(n^{2.373})$ using the best algorithms available [5]. However, when working with datasets containing 10^5 nodes, which is not that large in graph studies, computing PageRank or the heat kernel through matrix multiplication requires at least 10^{11} operations, requiring significant computing power.
The authors suggest an $O(N)$ approximation method for computing the diffusion matrix, but they do not provide any details about this method in the article.

2.7 Limitations

The authors caution that GDC relies on the homophily assumption, meaning that similar entities tend to group together. In many instances, data adheres to this principle. For example, in social media networks, individuals with similar political beliefs, such as conservatives or liberals, are more likely to interact with like-minded people, creating "echo chambers" or "filter bubbles" [1]. Nevertheless, in some case we might observe heterophily. An example in graphs can be observed in a collaboration network among researchers. Nodes represent individual researchers, while edges signify interdisciplinary collaborations on research projects or co-authorship on scientific papers. Researchers from diverse fields, such as computer science, biology, and physics, may come together to tackle complex, multidisciplinary problems. In this case, the CDG may not perform well.

3 Implementation

3.1 Our code

The code for our implementation can be found in this Github repository. Although the GDC method is now implemented as a transformation in Pytorch-geometric[2], we chose to reimplement GNNs and GDC from scratch to gain a deeper understanding of the techniques. The purpose of this implementation is to compare the different transition matrix and determine which one performs better for GNNs on a particular dataset.

1. GCN with transition matrix equal to L_{sym}
2. GCN with transition matrix equal to L_{rw}
3. GDN with transition matrix equal pagerank
4. GDN with transition matrix equal ppr

3.2 dataset

We have selected the 'Cora' dataset for our study, which consists of 2,708 scientific publications categorized into one of seven classes and a citation network of 5,429 links. Each publication is represented by a 0/1-valued word vector based on the presence or absence of words from a 1,433-word dictionary. The objective is to predict a paper's subject using its words and citation network. This is then a multi-class problem. This dataset was also used by the authors of [2] to evaluate their method.

Initially, we considered the 'ogbn-arxiv' dataset, representing the citation network between all Computer Science (CS) arXiv papers indexed by MAG. However, this dataset is considerably larger, with around 150,000 nodes, and posed computational challenges when calculating heat and PPR. While processing the dataset as a sparse matrix worked, calculating the heat and PPR required inverting very large matrices and performing matrix multiplications. Since we wanted to use our custom-coded method, which is the exact method and not the approximated one not detailed in [2], we then decided to use the smaller 'Cora' dataset.

3.3 Model details

The model we used in this node classification task is the GNN presented in the first section of this report, with the following hyperparameters:

hidden layers	hidden dimension	dropout	learning rate	batch normalization
2	264	0.3	3.10^{-4}	Yes

We experimented with various hyperparameters for the different transition matrices, but the results were not significantly different, so we chose to use the same set of hyperparameters. The evaluation metric is accuracy, which assesses the number of nodes correctly classified. Our model is trained using negative log-likelihood loss for this multiclass problem.

[2] `https://pytorch-geometric.readthedocs.io/en/latest/modules/transforms.html#torch_geometric.transforms.GDC`

3.4 Results

Transition matrix	epochs	Train accuracy	test accuracy
L_{sym}	25	95%	60%
L_{rw}	25	94%	55%
Heat Kernel	15	95%	90%
personalized PageRank (PPR)	15	94%	90%

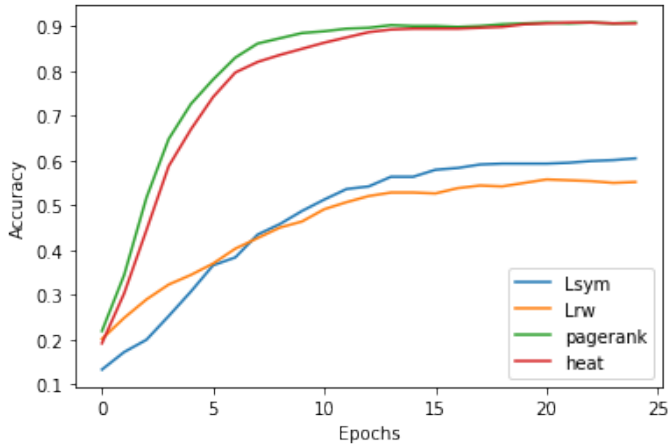

Figure 2: Performance by transition matrix

The results observed are quite impressive. Both Heat and PPR significantly outperform the traditional GCN with Laplacian matrices in terms of training speed and performance. Our results deviate slightly from those in the article [2] (accuracy of Heat : 83%, and accuracy of PPR : 84%), even though we are using the same "CORA" dataset, due to the use of a different train-test split. Consequently, these findings suggest that Heat and PPR provide better performance than the standalone GCN.

A possible future improvement would be to perform cross-validation or a random search on the hyperparameters for each transition matrix.

4 Conclusion

This article is quite intriguing, presenting a notable technical advancement in graph learning. With 340 citations, it highlights the significant improvement of algorithm performance on graphs. Despite its advantages, there are some limitations to this method. Like graphs in general, it has a high memory complexity, and we had to use a small dataset of 2700 nodes to run the method due to this. We successfully replicated the results on this dataset and achieved robust outcomes in a node classification task.
The presented method cannot perform at scale with large datasets, since the calculations cannot be executed entirely in a sparse matrix, necessitating a dense format. However, it is possible to approximate the method for handling large datasets, but it is not presented in the article. Furthermore, the method assumes homophily, which is not always valid, resulting in another limitation. Additionally, graphs are not widely used compared to text format or images.
Despite the implementation section being limited, the algorithm's theoretical aspect is quite lucid, and the article offers comprehensive coverage by approaching the subject from various angles, even though it may be challenging to grasp the entire concept.

References

[1] Nabeel Gillani, Ann Yuan, Martin Saveski, Soroush Vosoughi, and Deb Roy. Me, my echo chamber, and i: introspection on social media polarization. 2018.

[2] Gasteiger Johannes, Weißenberger Stefan, and Stephan Günnemann. Diffusion improves graph learning. 2019.

[3] Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete structures. 2002.

[4] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. 1998.

[5] Virginia Vassilevska Williams. Multiplying matrices in $o(n^{2.373})$ time. 2014.

[6] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. 2019.