CPSC 483/583: Deep Learning on Graph-Structured Data Yale University

Assignment 3

Out: October 11, 2024 Due: October 25, 2024

General Instructions

These questions require thought, but do not require long answers. Please be as concise as possible. You are allowed to take a maximum of 1 late period (see the course website or slides about the definition of a late period).

Submission instructions: You should submit your answers in a *single* PDF file. LaTeX is highly preferred due to the need of formatting equations.

Submitting answers: Prepare answers to your homework in a *single* PDF file. Make sure that the answer to each sub-question is on a *separate page*. The number of the question should be at the top of each page.

Honor Code: When submitting the assignment, you agree to adhere to the Yale Honor Code. Please read carefully to understand what it entails!

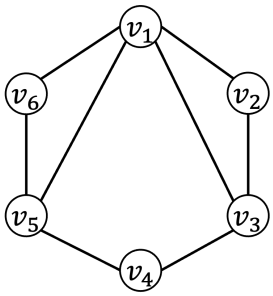

Figure 1: Graph 1

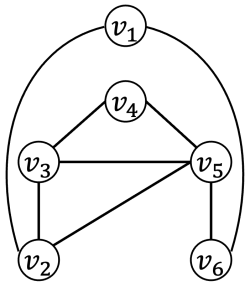

Figure 2: Graph 2

1 Graph Isomorphism

1. Graph neural networks update node embeddings through the computation graph defined by the neighborhood. Draw the computation graph of *Node 1* in *Graph 1* in Figure 1 for 2-layer GNNs (i.e., aggregate neighbor nodes twice).

   Note: you can draw in any way you like, take a picture or a screenshot to insert the figure into your submitted answer.

2. If every node in *Graph 1* has the same initial features, and no IDs will be seen by GNNs, which pairs of nodes cannot be distinguished by GNNs? Explain why.

   Is there any method to make the GNNs distinguish between these node pairs?

3. In graph theroy, graph G and graph H are **isomorphic** if there is a bijection between the vertex sets of G and H: $f : V(G) \to V(H)$ such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H.

   Are *Graph 1* in Figure 1 and *Graph 2* in Figure 2 isomorphic? If so, demonstrate the bijection function f between *Graph 1* and *Graph 2*. If not, prove it by finding a structure in one graph but is not present in the other. (**Hint:** use color refinement algorithm to extract the structure of K-hop neighborhood. Two isomorphic graphs should share the same K-hop neighboring information for any K.)

2 Contrastive Learning

Contrastive learning refers to a powerful class of embedding methods, and is widely used in self-supervised learning scenarios [[5], [6]]. The fundamental idea behind the constrastive learning paradigm is to encourage similar pairs of input samples to be represented close to each other, while dissimilar ones are pushed far apart. Contrastive learning can be applied in settings where it is possible to construct similar and dissimilar pairs in a training dataset (a process known as data

augmentation). For example, in computer vision, an image can be rotated to generate a 'similar' pair. For graphs, pairs can be existing nodes: often, graphs have a small proportion of positive pairs (ex. neighboring nodes) and a much larger proportion of negative pairs (ex. all pairs of nodes without links between them). Negative sampling is a common technique to improve training efficiency for large graphs, whereby a small subset of negative examples is randomly selected for each positive example. We will cover negative sampling in Homework 4.

1. Let's first apply the contrastive learning paradigm to a multi-way classification task: we have a list of input samples $\{x_i\}$, each with a corresponding label $\{y_i\}$ among k classes. We would like to learn a function f that encodes each input x_i into an embedding vector $f(x_i)$, such that data from the same class have similar embeddings and data from different classes have very different ones. This encourages higher quality embeddings that are more separable, thereby allowing for easier classification.

   The specific function f could be of any form (e.g. linear function or MLP); for now, we will keep it unspecified. A standard contrastive loss function is prescribed as

   $$\mathcal{L} = \sum_i \sum_j \mathbb{1}[y_i = y_j](1 - s(f(\mathbf{x}_i), f(\mathbf{x}_j))) + \mathbb{1}[y_i \neq y_j] \max(0, \epsilon - (1 - s(f(\mathbf{x}_i), f(\mathbf{x}_j)))) \quad (1)$$

   where $\mathbb{1}[x] = 1$ if x evaluates to true and 0 otherwise, and $s(\mathbf{z}_i, \mathbf{z}_j) = \frac{\mathbf{z}_i \cdot \mathbf{z}_j}{||\mathbf{z}_i|| \cdot ||\mathbf{z}_j||}$ is the cosine similarity metric. Note that $\epsilon > 0$. In a few sentences, explain how this loss function accomplishes the desired goal.

2. In the equation above, what is the significance of the hyperparameter ϵ?

3. To apply this paradigm to a graph setting, we need to first define what 'similar' means for graph learning.

   For now, let us first consider node-prediction tasks. The similarity function $s_{\mathcal{G}}(v_i, v_j)$ should have a high value for similar nodes, and a low value for dissimilar nodes. Here v_i and v_j refer to the two nodes of interest. Some examples include:

   - $s_{\mathcal{G}}(v_i, v_j) = \mathbf{A}_{ij}$, entries of the adjacency matrix
   - $s_{\mathcal{G}}(v_i, v_j) = \mathbf{A}^2_{ij} + \mathbf{A}_{ij}$, incorporating 2-hop connections.
   - $s_{\mathcal{G}}(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|}$, Jaccard's Coefficient, where $N(v_i)$ is the set of direct (1-hop) neighbors of vertex v_i.

   For the following graph (Figure 3), compute each of the above similarity measures for:

   (a) Vertex A and Vertex C

   (b) Vertex B and Vertex E

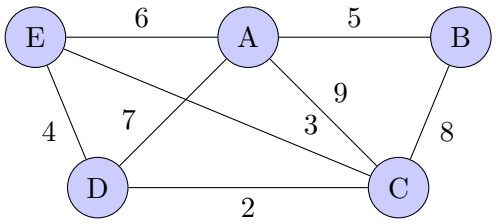

Figure 3: Example Graph for Similarity Metrics. Note: ignore the edge weights for the purposes of this exercise, although there are certainly similarity metrics that incorporate edge weight.

(c) Vertex C and Vertex D

4. The contrastive learning loss function from Eq. 1 uses class labels (y_i) to define positive and negative pairs. However, in unsupervised graph learning settings, one must use "self-supervised" metrics of similarity, such as those in part 3 of this question. Then, one must craft "similar" and "dissimilar" pairs of nodes. One possible method to do this is to first define a similarity threshold for $s_{\mathcal{G}}$, denoted as γ. Use γ to craft an unsupervised loss function for contrastive learning of node embeddings by modifying the loss function in Eq. 1.

5. Contrastive loss functions might help combat the phenomenon of oversmoothing in graph neural networks (ex. (Chen et al.)), wherein node embeddings become similar. Oversmoothing can result from excessively deep GNN architectures, or from inadequate loss functions. In a sentence or two, explain how the contrastive loss function you devised in the last part might mitigate oversmoothing of the second kind.

3 Generalized Graph Diffusion

Graph Diffusion Convolution (GDC) ([1], [3]) enhances Graph Neural Networks by leveraging methods like the heat kernel [4] and personalized PageRank [2] to form diffusion matrices, extending beyond immediate neighbors. Bridging spatial and spectral approaches, GDC outperforms traditional GCN message passing and is both computationally efficient and generalizable compared to spectral methods. GDC is proven to extract a graph's spectral information [3].

3.1 Diffusion Matrix

The diffusion matrix, S, can be characterized as a weighted summation across the powers of transition matrices. Mathematically, it is expressed as:

$$S = \sum_{k=0}^{\infty} \theta_k T_s^k \tag{2}$$

where θ_k is the weighting coefficient and T represents a transition matrix.
For the summation to converge, it must satisfy two key constraints:

a. The sum of the weights θ_k should be equal to one:

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

b. All eigenvalues of the transition matrix should be bounded between 0 and 1:

$$\forall i : \lambda_i \in [-1, 1]$$

Assume $T_s = AD^{-1}$, where D is the degree matrix and A is the adjacency matrix of the graph as a specific transition matrix that is used for diffusion.

1. Several series, such as power or geometric series, have been demonstrated to be effective in practice as coefficients θ for GDC (see [4], [2]). The Personalized Page Rank algorithm can be formulated as Graph Diffusion [3] that employs a geometric series of θ_k defined as

   $$\theta_k = \alpha(1-\alpha)^k,$$

   where α is the damping factor for random walks [2]. Provide a proof confirming that this θ_k meets the constraint a.

2. Demonstrate that T_s adheres to constraint b.

3. A column stochastic matrix is a real square matrix, with each column summing to 1. Establish that T_s is column stochastic.

4. Formally, A transition matrix is a square matrix used to describe the transitions between states in a stochastic process. For a system with n states, the transition matrix P will be $n \times n$ in size, where each entry P_{ij} represents the probability of transitioning from state i to state j in one step. Consider a probabilistic random walk as an stochastic process, i.e. a process of traversing a graph where at each stage an edge is chosen uniformly at random. Prove that $T_s = AD^{-1}$ is the transition matrix of this random walk.

3.2 Sparsification

Two primary methodologies are prevalent when it comes to calculating the diffusion matrix. The first involves determining the exact matrix using a closed-form solution. Notably, closed-form solutions often exist for the summation detailed in equation 2. To further explore two of the most frequently employed diffusion matrices, references [2] and [4] can be consulted. Alternatively, the matrix can be approximated by constraining k within the range $[0, k_{\max}]$. For the purposes of this section, we shall limit $k_{\max}$ to 2 for the sake of simplicity.

1. Given this constraint, compute the diffusion matrix for $\theta_k = \frac{e^{-1}}{k!}$, and $k_{\max} = 2$, which implies $k \in \{0, 1, 2\}$, using the simple graph depicted in figure 4. **Please consider the self-loops in matrices**.

   Hints:

   (a) $T^0 = I$

   (b) $0! = 1$

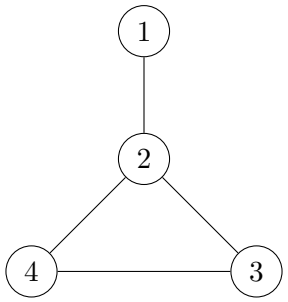

Figure 4: Simple graph of question 5

2. In real-world datasets, there's typically an average of 4 to 6 degrees of separation between any two nodes, leading to densely populated diffusion matrices. Hence, to decrease the complexity of calculating node embeddings in a GDC layer, it is essential to sparsify the diffusion matrix. A prevalent approach for sparsification that creates regular graphs and is amenable in many cases is *top-k*, which retains the top k elements of each column of the diffusion matrix. *top-k* coverts S to a regular matrix. Recompute the sparse diffusion matrix $\tilde{S}$ for the graph depicted in Figure 4 with $k = 3$.

   Hint: When sorting, break the ties in favor of the lower indices.

References

[1] James Atwood and Don Towsley. Diffusion-convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.

[2] Sergey Brin. The pagerank citation ranking: bringing order to the web. *Proceedings of ASIS, 1998*, 98:161–172, 1998.

[3] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *Advances in neural information processing systems*, 32, 2019.

[4] Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the 19th international conference on machine learning*, volume 2002, pages 315–322, 2002.

[5] Jiahong Liu, Menglin Yang, Min Zhou, Shanshan Feng, and Philippe Fournier-Viger. Enhancing hyperbolic graph embeddings via contrastive learning, 2022.

[6] Wei Xia, Tianxiu Wang, Quanxue Gao, Ming Yang, and Xinbo Gao. Graph embedding contrastive multi-modal representation learning for clustering. *IEEE Transactions on Image Processing*, 32:1170–1183, 2023.