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

Assignment 2

Out: September 20, 2024 Due: October 4, 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!

1 Invariance and Equivariance of GNNs

Permutation Matrix A permutation π of m element is defined by $\begin{pmatrix} 1 & 2 & \cdots & m \\ \pi(1) & \pi(2) & \cdots & \pi(m) \end{pmatrix}$, where the i-th element is permuted to $\pi(i)$. The permutation matrix $P_\pi = (p_{ij})$ is defined by $p_{ij} = 1$ if $j = \pi(i)$ and $p_{ij} = 0$ otherwise. For example, the permutation matrix P_π corresponding to the permutation $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 3 & 1 \end{pmatrix}$ is $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0t \end{bmatrix}$.

An important property of permutation matrices is that they are orthogonal matrices (i.e., $P_\pi P_\pi^\top = I$). Therefore, we have $P_\pi^{-1} = p_\pi^\top$.

Invariance and Equivariance Consider $X \in \mathbb{R}^{n\times d}$, a function $f(X)$ is **permutation invariant** if, for all permutation matrices $P \in \mathbb{R}^{n\times n}$: $f(PX) = f(X)$ holds. Accordingly, we say that $f(X)$ is **permutation equivariant** if, for all permutation matrices $P \in \mathbb{R}^{n\times n}$: $f(PX) = Pf(X)$ holds.

1. If there is a permutation P applied to the nodes, the original features $X \in \mathbb{R}^{n\times d}$ is changed to $PX \in \mathbb{R}^{n\times d}$, where n and d are the number of nodes and feature dimensions. Node permutations also accordingly act on the edges. Write out how the adjacency matrix $A \in \mathbb{R}^{n\times n}$ changes accordingly after node permutation.

2. Consider the message passing function of a graph convolutional layer: $H^{(l+1)} = \sigma(AH^{(l)}W_l^\top)$, $W_l^\top$ is the weight matrix. Prove that a graph convolutional layer is permutation equivariant.

3. Prove that if each layer of the network is permutation equivariant, then the network as a whole is permutation equivariant. (**Hint:** consider each layer as a function, then, stacking layers is equivalent to function composition. Prove the resulting composite function is permutation equivariant)

2 Graph Algorithm and Message Passing

Let's explore the relation between message passing mechanism and graph algorithms, such as graph random walk and breadth-first search.

1. **Random Walk:** Message-passing GNNs can simulate the graph random walk algorithm, where the number of layers corresponds to the time step t. In the graph random walk setting, each node in the graph G has an embedding at each time step. Let $h_i^{(t)}$ denote the embedding of node i at time step t and $h_i^{(0)}$ is a random initial embedding. $\mathcal{N}(i)$ denotes the neighbor nodes of node i. Assume we are at node i at time step t. At the next time step $t+1$,

we move from the current node i to one of its neighbor nodes $j \in \mathcal{N}(i)$. The probability of moving to each neighbor node is equal, which is called **uniform random walk**. After each move, the embedding of nodes will change. The expectation of the embedding of the node we move to is exactly the embedding of node i in the next time step $t+1$. Therefore, $h_i^{(t+1)} = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} h_j^{(t)}$.

Let $H^{(t)} \in \mathbb{R}^{n \times d}$, where n is the number of nodes and d is the embedding dimension. The i-th row of $H^{(t)}$ is the embedding of node i, i.e., $h_i^{(t)}$. We have $H^{(t+1)} = PH^{(t)}$, where P is called *transition matrix*. Derive the *transition matrix* using the adjacency matrix A and degree matrix D of the graph G.

2. **Breadth-first Search (BFS):** In BFS setting, there are two types of nodes with 1-dimensional embedding in $\{0, 1\}$ for a graph G. Let $h_i^{(t)}$ denote the embedding of node i at time step t, and $\mathcal{N}(i)$ denotes the neighbor nodes of i. The first type of node is visited nodes with an embedding of 1. Others are unvisited nodes with an embedding of 0. Initially, all nodes have embedding 0, except a starting node with an embedding of 1. At every step, nodes that are connected to any visited nodes become visited and change its embedding from 0 to 1. Nodes not visited keep the embedding 0.

   Message-passing GNNs can learn BFS algorithm well, where the number of layers corresponds to the time step. Write out a message function and an aggregation function with the notations given in the previous paragraph to update the embedding of node i at time step t to simulate this BFS algorithm. (**Hint:** Firstly, write out the update function of $h_i^{(t+1)}$, using the embedding of neighbor nodes at time step t (i.e., $h_j^{(t)}$ for $j \in \mathcal{N}(i)$). Figure out the message terms passed by the neighbor nodes. You can use min, max, sum, or any other well-defined non-linear functions to realize the aggregation)

3 Manifold Learning Using Laplacian Eigenvalues

An especially interesting application of graph learning can be found in a seemingly-unrelated domain: data analysis through **manifold learning**. High-dimensional data often lie on a low-dimensional surface called a *manifold*. An example of such data is shown in Figure 1: 3-dimensional data is plotted in a 3-D plot, with coloring used to highlight the shape of the roll. Oftentimes, as is the case in this example, this manifold is non-linear, and analyzing this data is often not best accomplished using linear dimensionality reduction algorithms (such as PCA), since they cannot account for the shape of the manifold.

However, such non-linear manifolds can be captured by embedding each datapoint **non-linearly** into a low-dimensional subspace via $f : \mathbb{R}^m \to \mathbb{R}^n$ with $n < m$, mapping each datapoint $f(\vec{x}_i) = \vec{h}_i$. To capture the manifold, we desire that the points close in the embedding space $\mathbb{R}^n$ will be close in the manifold, but not necessarily close in the Euclidean space of the original dataset. For example,

in Figure 1, even though the magenta and cyan colored data is close in Euclidean space, they are far apart along the manifold, since the cyan points neighbor the violet and green points, not the magenta ones. There exist many such non-linear 'manifold learning' methods; in this problem, we will focus on the longstanding, relatively straightforward method of Laplacian Eigenvalues, first introduced by Belkin & Niyogi here. Reading this paper is not required for this problem, but feel free to take a look if you are curious!

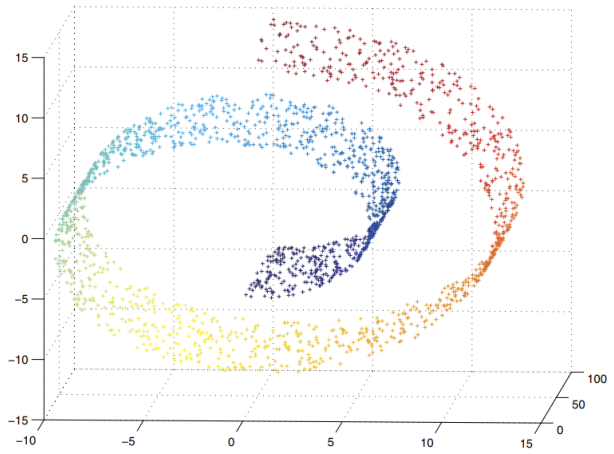

Figure 1: Swiss Roll Manifold; the colors represent the position along the manifold

We begin by defining a **weighted undirected graph** $G(V, E)$ with a vertex set V including l data points $\vec{x}_i \in \mathbb{R}^m$ for $i = 1, \cdots, l$. For each vertex (data point) $\vec{x}_i$ in G, we connect it to its N-nearest neighbors $\vec{x}_j, \ldots$ (where N is a hyper-parameter) in Euclidean space via an edge $(i, j) \in E$. The weight of the edge connecting node i to node j is given by a "Gaussian kernel": $W_{ij} = W_{ji} = \exp(-\frac{||\vec{x}_i - \vec{x}_j||_2^2}{2t^2})$, where $\vec{x}_i$ represents the vector for the data point i and $||\vec{x}_i - \vec{x}_j||_2^2$ represents the square of the l_2-norm (Euclidean distance) between data points $\vec{x}_i$ and $\vec{x}_j$. t is a hyper-parameter. Note that $W_{ij} = 0$ if i and j are not connected and $W_{ii} = 0$ for any i.

1. We first consider a 1-D embedding space; that is, $n = 1$. To ensure that data points that are close on the manifold are also close in the embedding space $\mathbb{R}$, the function f is designed to minimize the following loss function:

$$\mathcal{L} = \sum_{i \in V} \sum_{j \neq i \in V} W_{ij} (f(\vec{x}_i) - f(\vec{x}_j))^2$$

Question: In one or two sentences, explain how this loss function achieves the desired goal of manifold learning as stipulated above.

2. We define the graph Laplacian as $L = D - W$. W is the weight matrix derived from "Gaussian kernel" with $W_{ij} = W_{ji} = \exp(-\frac{||\vec{x}_i - \vec{x}_j||_2^2}{2t^2})$. $D = \text{diag}(d_1, d_2, \cdots, d_l)$ is a diagonal matrix where the i-th diagonal element $d_i = \sum_j W_{ij}$.

   Question: Show that the loss function described above can also be written as $\mathcal{L} = 2\mathbf{f}^T L\mathbf{f}$ where $\mathbf{f}$ is the vector of $f(\vec{x}_i)$ for all $i \in V$.

3. A quick linear algebra interlude. For any matrix M, we call $\vec{v}_i$ an eigenvector of M if $M\vec{v}_i = \lambda_i \vec{v}_i$, and we call λ_i its corresponding eigenvalue. Conceptually, M acts on $\vec{v}_i$ to scale it by a factor of λ_i, without altering its direction.

   Question: Show that the vector $\vec{1}$ consisting of all 1s (of dimension equal to $|V| = l$) is an eigenvector of the Laplacian $L = D - W$ and find the corresponding eigenvalue.

4. As discussed in the earlier problem on the Laplacian, the graph Laplacian L applied to a function g via pre-multiplication, Lg, measures the discrepancy between the value of g at a node v_i and the values at its neighboring nodes v_j, where $(i, j) \in E$. We are now interested in understanding why this embedding method is called Laplacian eigenvalues. Specifically, we aim to minimize the function $\mathcal{L} = \mathbf{f}^T L\mathbf{f}$. To ensure that the solution is normalized, *i.e.,* $\mathbf{f}^T\mathbf{f} = I$, we use the method of Langrage multipliers to incorporate the constraint into the loss function. This leads to the problem of finding f that achieves the stationary point of $\mathcal{L} = \mathbf{f}^T L\mathbf{f} - \lambda(\mathbf{f}^T\mathbf{f} - I)$, where λ is called the Lagrange multiplier.

   Question: By setting the derivative of the loss with respect to $\mathbf{f}$ to 0, derive the equation that expresses f as an eigenvector of L.

5. We now consider higher-dimensional embeddings for $n \geq 1$. To achieve this, we define multiple functions $f_1, \ldots, f_n$ and aim to minimize the loss function for each f_k as follows:

$$f_k = \arg\min_{f_k} \mathcal{L} = \mathbf{f_k}^T L\mathbf{f_k} - \lambda(\mathbf{f_k}^T\mathbf{f_k} - I) \text{ for } 1 \leq k \leq n.$$

   To ensure that these functions f_k represent distinct dimensions of the manifold and are not identical, we impose an additional constraint that the vectors $f_1, f_2, \ldots, f_n$ must be linearly independent. (A set of vectors is linearly independent if the only solution to $x_1 f_1 + x_2 f_2 + \cdots + x_n f_n = 0$ is $x_1 = x_2 = \cdots = x_n = 0$; otherwise, they are linearly dependent.)

   Question: Explain why the solution $f_k(\cdot)$ satisfies that $\mathbf{f_k}$ is the k-th smallest nonzero eigenvectors of the Laplacian L, where the smallest eigenvalue is 0.