EECS 182 Deep Neural Networks
Spring 2023 Anant Sahai

Homework 5

This homework is due on Friday, March 3, 2023, at 10:59PM.

1. Directed and Undirected Graphs

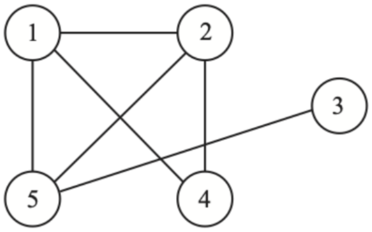

Figure 1: Simple Undirected Graph

Figure 1 shows a simple undirected graph whose adjacency matrices we want to make sure you can write down. Generally, an unnormalized adjacency matrix between the nodes of a directed or undirected graph is given by:

$$A_{i,j} = \begin{cases} 1 : \text{if there is an edge between node i and node j,} \\ 0 : \text{otherwise.} \end{cases} \tag{1}$$

This will be a symmetric matrix for undirected graphs. For a directed graph, we have:

$$A_{i,j} = \begin{cases} 1 : \text{if there is an edge from node i to node j,} \\ 0 : \text{otherwise.} \end{cases} \tag{2}$$

This need not to be symmetric for a directed graph, and is in fact typically not a symmetric matrix when we are thinking about directed graphs (otherwise, we'd probably be thinking of them as undirected graphs).

Similarly, the degree matrix of an undirected graph is a diagonal matrix that contains information about the degree of each vertex. In other words, it contains the number of edges attached to each vertex and it is given by:

$$D_{i,j} = \begin{cases} deg(v_i) : \text{if i == j,} \\ 0 : \text{otherwise.} \end{cases} \tag{3}$$

where the degree $\deg(v_i)$ of a vertex counts the number of times an edge terminates at that vertex.

For directed graphs, the degree matrix could be *In-Degree* when we count the number of edges coming into a particular node and *Out-Degree* when we count the number of edges going out of the node. We'll use the terms in-degree matrix or out-degree matrix to make it clear which one we are invoking.

Sometimes, imbalanced weights may undesirably affect the matrix spectrum (eigenvalues and eigenvectors). This occurs when a vertex with a large degree results in a large diagonal entry in the Laplacian matrix dominating the matrix properties. To solve that issue, a normalization scheme is applied which aims to make the influence of such vertices more equal to that of other vertices, by dividing the entries of the Adjacency matrix by the vertex degrees.

In that sense, a normalized adjacency matrix is given by:

$$A^{Normalized} = AD^{-1} \tag{4}$$

and a symmetrically normalized adjacency matrix is given by

$$A^{SymNorm} = D^{-1/2}AD^{-1/2} \tag{5}$$

Additionally, the Laplacian matrix relates many useful properties of a graph. In fact, the spectral decomposition of the Laplacian matrix of a graph allows for the construction of low-dimensional embeddings that appear in many machine learning applications. In other words, there is a relation between the properties of a graph and the spectra (eigenvalues and eigenvectors) of matrices associated with the graph, such as its adjacency matrix or Laplacian matrix.

Given a simple graph G with n vertices $v_1, ..., v_n$, its unnormalized Laplacian matrix $L_{n\times n}$ is defined element-wise as:

$$L_{i,j} = \begin{cases} deg(v_i) : \text{if i == j}, \\ -1 : \text{if i != j and } v_i \text{ is adjacent to } v_j, \\ 0 : \text{otherwise.} \end{cases} \tag{6}$$

or equivalently by the matrix:

$$L = D - A \tag{7}$$

where D is the degree matrix and A is the adjacency matrix of the graph.

We could also compute the symmetrically normalized Laplacian which is inherited from the adjacency matrix normalization scheme as shown below:

$$L^{SymNorm} = I - A^{SymNorm} \tag{8}$$

where I is the identity matrix, A is the unnormalized adjacency matrix, and L is the unnormalized Laplacian.

(a) **Show that $L^{SymNorm}$ could also be written as**:

$$L^{SymNorm} = D^{-1/2}LD^{-1/2} \tag{9}$$

where D is the dregree matrix, and L is the unnormalized Laplacian.

Solution:

$$\begin{aligned} L^{SymNorm} &= I - A^{SymNorm} \\ &= I - D^{-1/2}AD^{-1/2} \\ &= DD^{-1} - D^{-1/2}AD^{-1/2} \\ &= D^{-1/2}DD^{-1/2} - D^{-1/2}AD^{-1/2} \\ &= D^{-1/2}(D-A)D^{-1/2} \\ &= D^{-1/2}LD^{-1/2} \end{aligned} \tag{10}$$

(b) **Write the unnormalized adjacency A, the degree matrix, D, and the symmetrically normalized adjacency matrix, $A^{SymNorm}$, of the graph in Figure. 1.**

Solution:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

$$\text{A}^{SymNorm} = \begin{bmatrix} 0 & 1/3 & 0 & 1/\sqrt{6} & 1/3 \\ 1/3 & 0 & 0 & 1/\sqrt{6} & 1/3 \\ 0 & 0 & 0 & 0 & 1/\sqrt{3} \\ 1/\sqrt{6} & 1/\sqrt{6} & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/\sqrt{3} & 0 & 0 \end{bmatrix}$$

(c) **Write the symmetrically normalized Laplacian matrix of the graph in Figure. 1.**

Solution: $$L^{SymNorm} = \begin{bmatrix} 1 & -1/3 & 0 & -1/\sqrt{6} & -1/3 \\ -1/3 & 1 & 0 & -1/\sqrt{6} & -1/3 \\ 0 & 0 & 1 & 0 & -1/\sqrt{3} \\ -1/\sqrt{6} & -1/\sqrt{6} & 0 & 1 & 0 \\ -1/3 & -1/3 & -1/\sqrt{3} & 0 & 1 \end{bmatrix}$$

(d) **Compute A^2, A^3**

Solution: $$A^2 = \begin{bmatrix} 3 & 2 & 1 & 1 & 1 \\ 2 & 3 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 2 & 2 \\ 1 & 1 & 0 & 2 & 3 \end{bmatrix} A^3 = \begin{bmatrix} 4 & 5 & 1 & 5 & 6 \\ 5 & 4 & 1 & 5 & 6 \\ 1 & 1 & 0 & 2 & 3 \\ 5 & 5 & 2 & 2 & 2 \\ 6 & 6 & 3 & 2 & 2 \end{bmatrix}$$

We now want to estimate the traffic flow of inner downtown Berkeley and we know the road network shown below. The goal of the estimation is to estimate the traffic flow on eache road segment. The flow estimates should satisfy the conservation of vehicles exactly at each intersection as indicated by the arrows.

The intersections are labeled a to h. The road segments are labeled 1 to 22. The arrows indicate the direction of traffic.

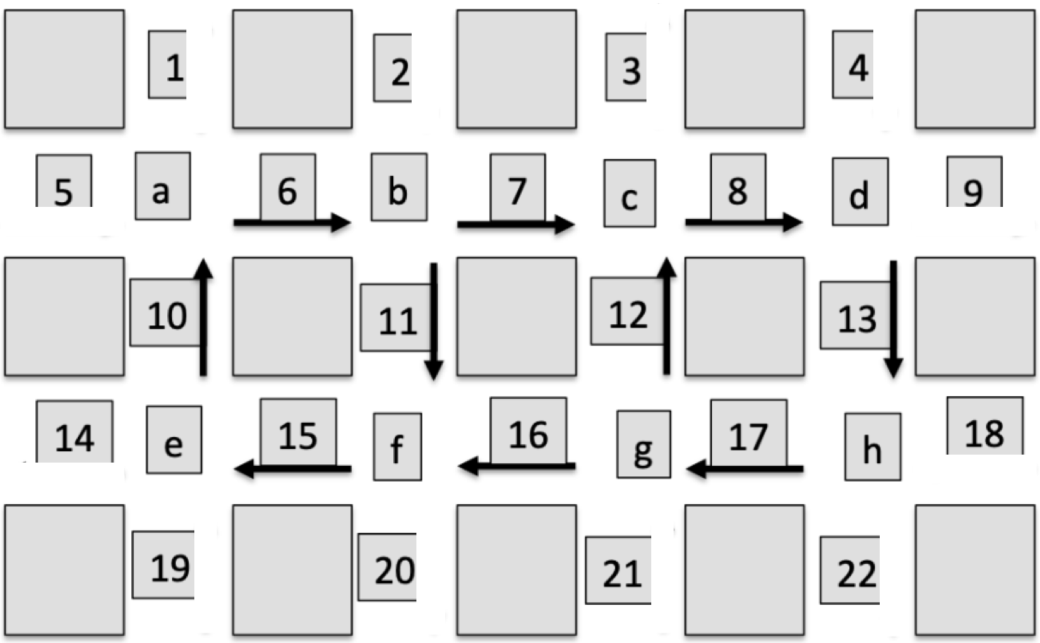

Figure 2: Simple Directed Graph

Hint: think about the best way to represent the road network in terms of matrices, vectors, etc.

(e) **Write the unnormalized adjacency matrix of the graph in Figure 2.**

Solution: We use the 8 intersections as nodes of the graph. This gives us an 8×8 adjacency matrix. Since the traffic flow is directed, it is a directed graph and the adjacency matrix is not symmetric.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

(f) **Write the In-degree D_{in} and Out-degree D_{out} matrix of the graph in Figure. 2.**

Solution: $$D_{in} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} D_{out} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(g) **Write both of the symmetrically normalized In-degree and Out-degree Laplacian matrix of the**

graph in Figure. 2.

$$L_{in}^{SymNorm} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1/\sqrt{2} & 0 & 0 & -1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 1 & -1/\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1/\sqrt{2} & 1 & 0 & 0 \\ 0 & 0 & -1/\sqrt{2} & 0 & 0 & -1/\sqrt{2} & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

Solution:

$$L_{out}^{SymNorm} = \begin{bmatrix} 1 & -1/\sqrt{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1/\sqrt{2} & 0 & 0 & -1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1/\sqrt{2} & 0 & 0 & -1/\sqrt{2} & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1/\sqrt{2} & 1 \end{bmatrix}$$

(h) ***[Optional]*** It is good to read https://arxiv.org/pdf/1609.02907.pdf and https://distill.pub/2021/understanding-gnns/ to learn about the importance of the Adjacency and Laplacian matrices in graph representation.

2. Graph Dynamics

Some graph neural network methods operate on the full adjacency matrix. Others, such as those discussed here https://distill.pub/2021/gnn-intro/, at each layer apply the same local operation to each node based on inputs from its neighbors.

This problem is designed to:

- show connections between these methods.
- show that for a positive integer k, the matrix A^k has an interesting interpretation. That is, the entry in row i and column j gives the number of walks of length k (i.e., a collection of k edges) leading from vertex i to vertex j.

To do this, let's consider a very simple deep linear network that is built on an underlying graph with n vertices. In the 0-th layer, each node has a single input with weight 1 that is fed a one-hot encoding of its own identity — so node i in the graph has a direct input which is an $n-$dimensional vector that has a 1 in position i and 0s in all other positions. You can view these as n channels if you want.

The weights connecting node i in layer k to node j in layer $k+1$ are simply 1 if vertices i and j are connected in the underlying graph and are 0 if those vertices are not connected in the underlying graph. At each layer, the operation at each node is simply to sum up the weighted sum of its inputs and to output the resulting n-dim vector to the next layer. You can think of these as being depth-wise operations if you'd like.

(a) Let A be the $n \times n$ size adjacency matrix for the underlying graph where the entry $A_{i,j} = 1$ if vertices i and j are connected in the graph and 0 otherwise. **Write the output of the j-th node at layer k in this network in terms of the matrix A.**

(Hint: This output is an n-dimensional vector since there are n output channels at each layer.)

Solution: A_j^k

Explanation: Think of the initial graph as a matrix of length n, where every node is a row. The initial graph is just the identity.

At each timestep, we can compute the next timestep of the graph by taking $G^{t+1} = AG_t$. This works b/c every row j in G^{t+1} is produced by summing the rows of G_t specified by the 1s in A_j - i.e. the rows corresponding to the neighbors of node j.

If we do this repeatedly, we get that the value at node j is $(A^k G_0)_j = A_j^k$.

(b) Here is some helpful notation: Let $V(i)$ be the set of vertices that are connected to vertex i in the graph. Let $L_k(i,j)$ be the number of distinct paths that go from vertex i to vertex j in the graph where the number of edges traversed in the path is exactly k. Recall that a path from i to j in a graph is a sequence of vertices that starts with i, ends with j, and for which every successive vertex in the sequence is connected by an edge in the graph. The length of the path is 1 less than the number of vertices in the corresponding sequence. **Show that the i-th output of node j at layer k in the network above is the count of how many paths there are from i to j of length k**, where by convention there is exactly 1 path of length 0 that starts at each node and ends up at itself.

(Hint: Can applying induction on k help?)

Solution: **Proof**: We can prove the result by induction on k. For k = 1, the result follows from the very definition of A. Let $L_k(i,j)$ denote the number of paths of length k between nodes i and j, and assume that the result we wish to prove is true for some given $h \geq 1$, so that $L_h(i,j) = [A_h]_{i,j}$.

We next prove that it must also hold that $L_{h+1}(i,j) = [A_{k+1}]_{i,j}$, thus proving by inductive argument that $L_k(i,j) = [A_k]_{i,j}$ for all $k \geq 1$.

Indeed, to go from a node i to a node j with a walk of length h + 1, one needs first reach, with a walk of length h, a node l linked to j by an edge. Thus:

$$L_{h+1}(i,j) = \Sigma_{l \in V(j)} L_h(i,l) \tag{11}$$

where V(j) is the neighbor set of j, which is the set of nodes connected to the j-th node, that is, nodes l such that $A_{l,j} \neq 0$. Thus:

$$L_{h+1}(i,j) = \Sigma_{l=1}^{n} L_h(i,l) A_{l,j} \tag{12}$$

But we assumed that $L_h(i,j) = [A_h]_{i,j}$, hence the previous equation can be written as:

$$L_{h+1}(i,j) = \Sigma_{l=1}^{n} [A_h]_{i,l} A_{l,j} \tag{13}$$

In the above we recognize the (i,j)-th element of the product $A^h A$ = A_{h+1}, which proves that $L_{h+1}(i,j)$ = $[A_{h+1}]_{i,j}$, and hence concludes the inductive proof.

(c) The structure of the neural network in this problem is compatible with a straightforward linear graph neural network since the operations done (just summing) are locally permutation-invariant at the level of each node and can be viewed as essentially doing the exact same thing at each vertex in the graph based on inputs coming from its neighbors. This is called "aggregation" in the language of graph neural nets. In the case of the computations in previous parts, **what is the update function that takes the aggregated inputs from neighbors and results in the output for this node?**

Solution: If we represent the graph as a matrix with a node value in each row, the update function is to multiply by A.

If we represent the graph as a set of nodes, the update function is
$v_j = \sum_{i \in V(j)} \mathrm{v}_i$

(d) The simple GNN described in the previous parts counts paths in the graph. If we were to replace sum aggregation with max aggregation, **what is the interpretation of the outputs of node j at layer k?**

Solution: It is 1 if there is a path from i to j of length k and 0 otherwise.

3. The power of the graph perspective in clustering (Coding)

Implement all the TODOs in the `hw5_graph_clustering.ipynb` (colab link) notebook. **Answer the written questions below and include your completed notebook with your submission**.

(a) We used the KMeans algorithm implementation of sklearn, and showed our attempt to cluster this dataset into 3 classes. **Comment on the output the KMeans algorithm? Did it work? If so explain why, if not, explain not.**

Solution: As could be seen in the plots, it did not work. Note that the provided dataset is not linearly separable and the Kmeans clustering algorithm minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances.

This algorithm works by finding centroids and looks at points around each centroid inside a given radius to determine which points correspond to which cluster.

Even if we try giving the kmeans algorithm the correct means of each cluster directly, it still will classify the points in the wrong way because as given, they have the wrong embedding. In that sense, each point votes for its cluster in an isolated way.

Kmeans works using centroids which is the representation of the data center and each point is assigned to the nearest centroid.

(b) As given, the data points in our dataset are represented simply with their 2D Cartesian coordinates. Let's now interpret every single point as a node in a graph. Our goal is to find a way to relate every node in the graph in such way that the points that are closer together and points that are far apart maintain that relationship explicitly.

That is, we will choose to look at every point in the dataset as a vertex in a graph where the edge connection between two vertexes is determined by the weighted distances between them. **Write a function that takes in the input dataset and some coefficient gamma and returns the adjacency matrix A. Is this a directed or an undirected graph?**

$$A_{i,j} = e^{-\gamma ||x_i - x_j||^2} \tag{14}$$

where x_i and x_j represent each point in the provided dataset, γ is positive. You may find the *distance* module from *scipy.spatial* useful.

Solution: undirected; See solution notebook.

(c) The degree matrix of an undirected graph is a diagonal matrix that contains information about the degree of each vertex. In other words, it contains the number of edges attached to each vertex and it is given by Eq (4) in problem 3. Note that in the traditional definition of the adjacency matrix, this boils down to the diagonal matrix in which elements along the diagonals are the column-wise sum of the elements in the adjacency matrix. Using the same idea, **write a function that takes in the adjacency matrix as an argument and returns the degree matrix.**

Solution: See notebook solution for the function.

(d) Using γ = 7.5, **compute the adjacency matrix A, degree matrix D and the symmetrically normalized adjacency matrix matrix M,**

$$M = A^{SymNorm} = D^{-1/2}AD^{-1/2} \tag{15}$$

Solution: See solution notebook.

Note that another interpretation of the matrix M is that it shows the probability of moving/jumping from one node to another.

(e) Applying SVD decomposition on M, **write a function that selects the top 3 vectors (corresponding to the highest singular values) in the matrix U and performs the same KMeans clustering used above on them ; show the plots. What do you observe? Did it work? If so explain why, if not, explain not.**

Intuition: By selecting the top 3 vectors of the U matrix, we are selecting a new representation of the data points which could be seen as a construction of a low dimension embedding of the data points as mentioned in problem 3.

Solution: While it did better than the first try, it is still not separating the upper 2 clusters properly. Recall that when we write $M = U\Sigma \mathrm{V}^T$ using the SVD decomposition for some matrix M, we are simply mapping from a set of ortho-normal vectors V_i to a different set of ortho-normal vectors U_i. That is, every U_i is scaled by σ_i and rotated in such way that orthogonality is maintained. In that sense, it also captures directions of the variances from high to low.

(f) Now let's think of the symmetrically normalized adjacency matrix obtained above as the transition matrix in of a Markov Chain. That is, it represents the probability of jumping from one node to another. In order to fully interpret M in such way, it needs to be a proper stochastic matrix which means that the sum of the elements in each column must add up to 1. **Write a function that takes in the matrix M and returns M_{stoch}, the stochastic version of M; compute the stochastic matrix.**

Solution: See solution notebook.

Using SVD decomposition on the newly obtained stochastic matrix M_{stoch}, **use your function in part (e) to select the top 3 vectors of the matrix U_{stoch} and perform the same KMeans clustering used above on them and show the plots. What do you observe? Did it work?**

Solution: See solution notebook. It works. M_{stoch} can be viewed as the transition probabilities between the data points in a random walk on the graph. The top eigenvectors of this matrix capture the dominant patterns of the random walk, which can be used to cluster the data points. By taking the top eigenvectors, we can identify the most significant patterns of connectivity in the graph and use them to group similar data points together.

4. Graph Neural Networks

For an undirected graph with no labels on edges, the function that we compute at each layer of a Graph Neural Network must respect certain properties so that the same function (with weight-sharing) can be used at different nodes in the graph. Let's focus on a single particular "layer" ℓ. For a given node i in the graph, let $\mathbf{s}_i^{\ell-1}$ be the self-message (i.e. the state computed at the previous layer for this node) for this node from the preceeding layer, while the preceeding layer messages from the n_i neighbors of node i are denoted by $\mathbf{m}_{i,j}^{\ell-1}$ where j ranges from 1 to n_i. We will use w with subscripts and superscripts to denote learnable scalar weights. If there's no superscript, the weights are shared across layers. Assume that all dimensions work out.

(a) **Tell which of these are valid functions for this node's computation of the next self-message $\mathbf{s}_i^\ell$. For any choices that are not valid, briefly point out why.**

Note: we are *not* asking you to judge whether these are useful or will have well behaved gradients. Validity means that they respect the invariances and equivariances that we need to be able to deploy as a GNN on an undirected graph.

(i) $\mathbf{s}_i^\ell = w_1 \mathbf{s}_i^{\ell-1} + w_2 \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{m}_{i,j}^{\ell-1}$

Solution: This is valid because it is permutation invariant to the ordering of neighbors. This is the classic averaging form. Notice that a dependence on the number of neighbors is fine.

(ii) $\mathbf{s}_i^\ell = \max(w_1^\ell \mathbf{s}_i^{\ell-1}, w_2 \mathbf{m}_{i,1}^{\ell-1}, w_3 \mathbf{m}_{i,2}^{\ell-1}, \ldots, w_{n_i-1} \mathbf{m}_{i,n_i}^{\ell-1})$ where the $\max$ acts component-wise on the vectors.

Solution: This is invalid. Since different scalar weights are applied to different $\mathbf{m}$, it is not permutation invariant to the ordering of neighbors.

(iii) $\mathbf{s}_i^\ell = \max(w_1^\ell \mathbf{s}_i^{\ell-1}, w_2 \mathbf{m}_{i,1}^{\ell-1}, w_2 \mathbf{m}_{i,2}^{\ell-1}, \ldots, w_2 \mathbf{m}_{i,n_i}^{\ell-1})$ where the $\max$ acts component-wise on the vectors.

Solution: This is valid. Since the same weight w_2 is applied to all $\mathbf{m}$, it is permutation invariant to the ordering of neighbors. The max is another classic permutation-invariant operation.

(b) We are given the following simple graph on which we want to train a GNN. The goal is binary node classification (i.e. classifying the nodes as belonging to type 1 or 0) and we want to hold back nodes 1 and 4 to evaluate performance at the end while using the rest for training. We decide that the surrogate loss to be used for training is the average binary cross-entropy loss.

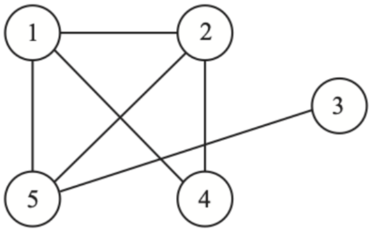

Figure 3: Simple Undirected Graph

nodes	1	2	3	4	5
y_i	0	1	1	1	0
$\hat{y}_i$	a	b	c	d	e

Table 1: y_i is the ground truth label, while $\hat{y}_i$ is the predicted probability of node i belonging to class 1 after training.

Table 1 gives you relevant information about the situation.

Compute the training loss at the end of training.

Remember that with n training points, the formula for average binary cross-entropy loss is

$$\frac{1}{n} \sum_x \left(y(x) \log \frac{1}{\hat{y}(x)} + (1 - y(x)) \log \left(\frac{1}{1 - \hat{y}(x)} \right) \right)$$

where the x in the sum ranges over the training points and $\hat{y}(x)$ is the network's predicted probability that the label for point x is 1.

Solution:
Since our testing nodes are nodes $\{1, 4\}$, a mask will be used to remove their contribution to the loss. We therefore get:

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{3}[y_2 \times \log(\hat{y}_2) + y_3 \times \log(\hat{y}_3) + y_5 \times \log(\hat{y}_5) + (1 - y_5) \times \log(1 - \hat{y}_5)] \tag{16}$$

$$= -\frac{1}{3}[(1. \times \log(b)) + (1. \times \log(c)) + (0 \times \log(e)) + \log(1 - e)] \tag{17}$$

$$= -\frac{1}{3}[\log(b) + \log(c) + \log(1 - e)] \tag{18}$$

$$= -\frac{1}{3}[\log\big(bc(1 - e)\big)] \tag{19}$$

(c) Suppose we decide to use the following update rule for the internal state of the nodes at layer ℓ.

$$\mathbf{s}_i^{\ell} = \mathbf{s}_i^{\ell-1} + W_1 \frac{\sum_{j=1}^{n_i} \tanh\left(W_2 \mathbf{m}_{i,j}^{\ell-1}\right)}{n_i} \tag{20}$$

where the $\tanh$ nonlinearity acts element-wise.
For a given node i in the graph, let $\mathbf{s}_i^{\ell-1}$ be the self-message for this node from the preceeding layer, while the preceeding layer messages from the n_i neighbors of node i are denoted by $\mathbf{m}_{i,j}^{\ell-1}$ where j ranges from 1 to n_i. We will use W with subscripts and superscripts to denote learnable weights in matrix form. If there's no superscript, the weights are shared across layers.

(i) **Which of the following design patterns does this update rule have?**

□ Residual connection

□ Batch normalization

Solution: This rule shows the residual connection pattern since the previous layer's state gets added in. This kind of residual connection enables gradients to flow back to earlier layers more easily. None of the other patterns are present.

(ii) **If the dimension of the state s is d-dimensional and W_2 has k rows, what are the dimensions of the matrix W_1?** **Solution:** W_1 needs to return something that can add to the state. This means that it needs d rows. It also has to be able to act on vectors that are k dimensional since W_2 has k rows and we say that the $\tanh$ operation acts element-wise. This means that W_1 needs k columns. Putting this together, the dimensions of W_1 are $d \times k$.

(iii) If we choose to use the state $\mathbf{s}_i^{\ell-1}$ itself as the message $\mathbf{m}^{\ell-1}$ going to all of node i's neighbors, **please write out the update rules corresponding to (20) giving $\mathbf{s}_i^{\ell}$ for the graph in Figure 3 for nodes $i = 2$ and $i = 3$ in terms of information from earlier layers.** Expand out all sums.

Solution: We can write the mean-pooling/ average pooling as in the the neighborhood N_i is of node i:
Therefore:
i = 2:

$$s_2^l = s_2^{l-1} + \frac{W_1}{3}(\tanh\left(W_2 s_1^{l-1}\right) + \tanh\left(W_2 s_4^{l-1}\right) + \tanh\left(W_2 s_5^{l-1}\right))$$

i = 3:

$$s_3^l = s_3^{l-1} + \frac{W_1}{1}(\tanh\left(W_2 s_5^{l-1}\right))$$

The above equations can be found by reducing the sum in equation 20 and applying the accurate neighbors from Figure 3.

5. Zachary's Karate Club (Coding)

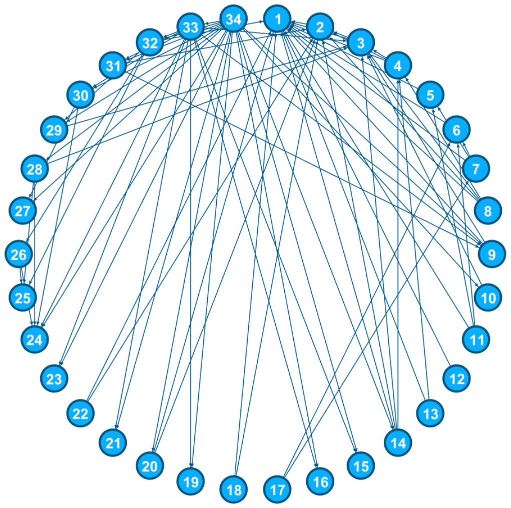

Figure 4: Zachary's Karate Club Graph

Zachary's Karate Club (ZKC) is a social network of a university karate club, described in the paper "An Information Flow Model for Conflict and Fission in Small Groups" by Wayne W. Zachary.

A social network captures 34 members of a karate club, documenting links between pairs of members who interacted outside the club.

During the study a conflict arose between the officer/ administrator ("John A") and the instructor "Mr. Hi", which led to the split of the club into two.

Half of the members formed a new club around Mr. Hi; members from the other part found a new instructor or gave up karate.

Based on collected data Zachary correctly assigned all but one member of the club to the groups they actually joined after the split. You could read more about it here `https://www.jstor.org/stable/3629752`, and here `https://commons.wikimedia.org/wiki/File:Social_Network_Model_of_Relationships_in_the_Karate_Club.png`

We will train a GNN to cluster people in the karate club in such that people who are more likely to associate with either the officer or Mr. Hi will be close together, while the distance beween the 2 classes will be far.

In the original paper titled "Semi-Supervised Classification with Graph Convolutional Networks" that can be found here `https://arxiv.org/pdf/1609.02907.pdf`, the authors framed this as a node-level classification problem on a graph. We will pretend that we only know the affiliation labels for some of the nodes (which we'll call our training set) and we'll predict the affiliation labels for the rest of the nodes (our test set).

Implement all the TODOs in `hw5_zkc.ipynb` (colab link) and include your notebook with your submission.

(a) Go through `q_zkc.ipynb`. We want our network to be aware of information about the nodes themselves instead of only the neighborhood, so we add self loops our adjacency matrix. The paper called this $\tilde{A}$. **Compute $\tilde{A}$ to add self loops to your adjacency matrix.**

Solution: See solution notebook.

(b) **Write a function that takes in $\tilde{A}$ as argument and returns the $\tilde{A}^{SymNorm}$ adjacency matrix**.

Solution: See solution notebook.

(c) The other input to our GNN is the graph node matrix X which contains node features. For simplicity, we set X to be the identity matrix because we don't have any node features in this example. **Generate the feature input matrix X.**

Solution: See solution notebook.

(d) We will now implement a single layer GNN. **Implement the forward and backward pass functions for `GNN_Layer` class.** Details can be found in the notebook.

Solution: See solution notebook.

(e) **Run the forward and backward passes and ensure the checks pass.**

Solution: See solution notebook.

(f) We are now ready to setup our classification network! **Use the GNN and Softmax layers to setup the network.**

Solution: See solution notebook.

(g) **Instantiate the GNN model with the correct input and output dimensions.**

Solution: See solution notebook.

(h) With the model, data and optimizer ready, **fill in the todos in the training loop function and train your model. Plot the clustered data.**

(i) **Explain why we obtain 100% on accuracy on our test set, yet we see in the plot that 2 samples seem to be misclassified.**

Solution: Those samples were part of the training set and were never used to evaluate the model at test time. All nodes used in the test set were correctly classified hence the 100% accuracy.

6. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework? Write it down here where you'll need to remember it for the self-grade form.**

Contributors:

- Jerome Quenum.
- Olivia Watkins.
- Anant Sahai.

- Anrui Gu.
- Matthew Lacayo.
- Past EECS 282 and 227 Staff.
- Romil Bhardwaj.