CS224W Homework 1

Due: Oct 16 11:59 p.m.

1 GNN Expressiveness (28 points)

For Q1.1, write down number of layers needed. For Q1.2, write down the transition matrix M and the limiting distribution r. For Q1.3 and 1.4, write down the transition matrix w.r.t A and D. For Q1.5, write down your proof in a few sentences (equations if necessary). For Q1.6, describe the message function, aggregate function, and update rule in a few sentences or equations.

Graph Neural Networks (GNNs) are a class of neural network architectures used for deep learning on graph-structured data. Broadly, GNNs aim to generate high-quality embeddings of nodes by iteratively aggregating feature information from local graph neighborhoods using neural networks; embeddings can then be used for recommendations, classification, link prediction, or other downstream tasks. Two important types of GNNs are GCNs (graph convolutional networks) and GraphSAGE (graph sampling and aggregation).

Let $G = (V, E)$ denote a graph with node feature vectors X_u for $u \in V$. To generate the embedding for a node u, we use the neighborhood of the node as the computation graph. At every layer l, for each pair of nodes $u \in V$ and its neighbor $v \in V$, we compute a message function via neural networks, and apply a convolutional operation that aggregates the messages from the node's local graph neighborhood (Figure 1.1), and updates the node's representation at the next layer. By repeating this process through K GNN layers, we capture feature and structural information from a node's local K-hop neighborhood. For each of the message computation, aggregation, and update functions, the learnable parameters are shared across all nodes in the same layer.

We initialize the feature vector for node X_u based on its individual node attributes. If we already have outside information about the nodes, we can embed that as a feature vector. Otherwise, we can use a constant feature (vector of 1) or the degree of the node as the feature vector.

These are the key steps in each layer of a GNN:

- **Message computation**: We use a neural network to learn a message function between nodes. For each pair of nodes u and its neighbor v, the neural network message function can be expressed as $M(h_u^k, h_v^k, e_{u,v})$. In GCN and GraphSAGE, this can simply be $\sigma(Wh_v + b)$, where W and b

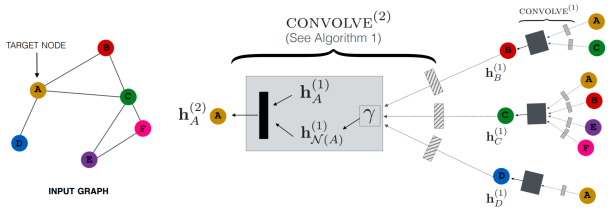


Figure 1.1: GNN architecture

are the weights and bias of a neural network linear layer. Here h_u^k refers to the hidden representation of node u at layer k, and $e_{u,v}$ denotes available information about the edge (u, v), like the edge weight or other features. For GCN and GraphSAGE, the neighbors of u are simply defined as nodes that are connected to u. However, many other variants of GNNs have different definitions of neighborhood.

- **Aggregation**: At each layer, we apply a function to aggregate information from all of the neighbors of each node. The aggregation function is usually permutation invariant, to reflect the fact that nodes' neighbors have no canonical ordering. In a GCN, the aggregation is done by a weighted sum, where the weight for aggregating from v to u corresponds to the (u, v) entry of the normalized adjacency matrix $D^{-1/2}AD^{-1/2}$.

- **Update**: We update the representation of a node based on the aggregated representation of the neighborhood. For example, in GCNs, a multi-layer perceptron (MLP) is used; Graph-SAGE combines a skip layer with the MLP.

- **Pooling**: The representation of an entire graph can be obtained by adding a pooling layer at the end. The simplest pooling methods are just taking the mean, max, or sum of all of the individual node representations. This is usually done for the purposes of graph classification.

We can formulate the Message computation, Aggregation, and Update steps for a GCN as a layer-wise propagation rule given by:

$$h^{k+1} = \sigma(D^{-1/2}AD^{-1/2}h^k W^k) \tag{1}$$

where h^k represents the matrix of activations in the k-th layer, $D^{-1/2}AD^{-1/2}$ is the normalized adjacency of graph G, W_k is a layer-specific learnable matrix, and σ is a non-linearity function. Dropout and other forms of regularization can also be used.

We provide the pseudo-code for GraphSAGE embedding generation below. This will also be relevant to the questions below.

Algorithm 1: Pseudo-code for forward propagation in GraphSAGE

Input : Graph $G(V, E)$; input features $\{x_v, \forall v \in V\}$; depth K;
non-linearity σ; weight matrices $\{W^k, \forall k \in [1, K]\}$;
neighborhood function $\mathcal{N} : v \rightarrow 2^V$;
aggregator functions $\{\text{AGGREGATE}_k, \forall k \in [1, K]\}$

Output: Vector representations z_v for all $v \in V$

$h_v^0 \leftarrow x_v, \forall v \in V$;
for $k = 1...K$ **do**
  for $v \in V$ **do**
    $h_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\})$ // aggregation
    $h_v^k \leftarrow \sigma\left(W^k \cdot \text{CONCAT}(h_v^{k-1}, h_{\mathcal{N}(v)}^k)\right)$ // MLP with skip connection
  $h_v^k \leftarrow h_v^k / \|h_v^k\|_2, \forall v \in V$ // update step
$z_v \leftarrow h_v^K, \forall v \in V$

In this question, we investigate the effect of the number of message passing layers on the expressive power of Graph Convolutional Networks. In neural networks, expressiveness refers to the set of functions (usually the loss function for classification or regression tasks) a neural network is able to compute, which depends on the structural properties of a neural network architecture.

.

1.1 Effect of Depth on Expressiveness (4 points)

Consider the following 2 graphs in figure 1.2, where all nodes have 1-dimensional initial feature vector $x = [1]$. We use a simplified version of GNN, with no non-linearity, no learned linear transformation, and sum aggregation. Specifically, at every layer, the embedding of node v is updated as the sum over the embeddings of its neighbors (N_v) and its current embedding h_v^t to get h_v^{t+1}. We run the GNN to compute node embeddings for the 2 red nodes respectively. Note that the 2 red nodes have different 5-hop neighborhood structure (note this is not the minimum number of hops for which the neighborhood structure of the 2 nodes differs). How many layers of message passing are needed so that these 2 nodes can be distinguished (i.e., have different GNN embeddings)? Explain your answer in a few sentences.

.

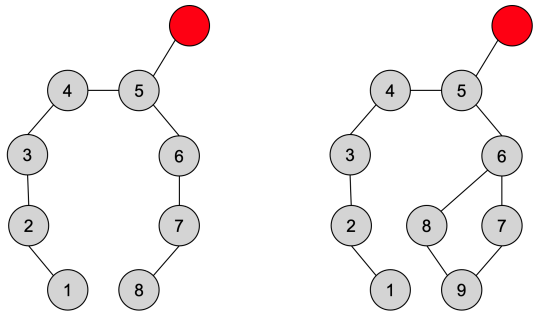


Figure 1.2: Figure for Question 1.1

★ Solution ★

- Layer 1: The red node receives information from node h_5^0 in both graphs; the value is the same.
- Layer 2: The red node needs information from the previous red node (the same), and the new node h_5^1 is synthesized from node (h_4^0, h_6^0) in both graphs, the value is still the same.
- Lyaer 3: The red node needs information from the previous red node (the same), and the new node h_5^2 is synthesized from node (h_4^1, h_6^1) in both graphs, the value is still the same. In both graphs, node h_4^1 is aggregated the same way, but node h_6^1 is different.

Therefore, three-layer communication is needed so that the two red nodes are different on the two graphs.

1.2 Random Walk Matrix (4 points)

Consider the graph shown below (figure 1.3).

1. Assume that the current distribution over nodes is $r = [0, 0, 1, 0]$, and after the random walk, the distribution is $M \cdot r$. What is the random walk transition matrix M, where each column of M corresponds with the node ID of the node that you are transitioning from?

2. What is the limiting distribution r, namely the eigenvector of M that has an eigenvalue of 1 ($r = Mr$)? Write your answer in fraction form or round it to the nearest thousandth place and in the following form, e.g. $[1.200, 0.111, 0.462, 0.000]$. Note that before reporting you should normalize r **Hint: r is a probability distribution representing the random walk probabilities for each node after a large number of timesteps**.

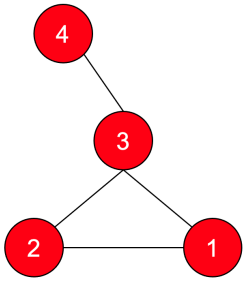


Figure 1.3: Figure for Question 1.2

★ Solution ★

1.

$$M = AD^{-1} = \begin{bmatrix} 0 & 1/2 & 1/3 & 0 \\ 1/2 & 0 & 1/3 & 0 \\ 1/2 & 1/2 & 0 & 1 \\ 0 & 0 & 1/3 & 0 \end{bmatrix}$$

2.

$$Mr = r \rightarrow (M - I)r = 0. \tag{2}$$

Result: $r = [0.250, 0.250, 0.375, 0.125]$

1.3 Relation to Random Walk (i) (4 points)

Let's explore the similarity between message passing and random walks. Let $h_i^{(l)}$ be the embedding of node i at layer l. Suppose that we are using a mean aggregator for message passing, and omit the learned linear transformation and non-linearity: $h_i^{(l+1)} = \frac{1}{|N_i|} \sum_{j \in N_i} h_j^{(l)}$. If we start at a node u and take a uniform random walk for 1 step, the expectation over the layer-l embeddings of nodes we can end up with is $h_u^{(l+1)}$, exactly the embedding of u in the next GNN layer. What is the transition matrix of the random walk? Describe the transition matrix using the adjacency matrix A, and degree matrix D, a diagonal matrix where $D_{i,i}$ is the degree of node i.

★ Solution ★

Transition matrix $P = D^{-1}A$, where A is the adjacency matrix, D^{-1} is a diagonal matrix with elements of $1/d_i$. When multiplying $D^{-1}A$, each element in row i of A is divided by the degree of that node (d_i). This matches the definition of the probability of moving from i to other places as $1/d_i$.

1.4 Relation to Random Walk (ii) (4 points)

Suppose that we add a skip connection to the aggregation from Question 1.3:

$$h_i^{(l+1)} = \frac{1}{2} h_i^{(l)} + \frac{1}{2} \frac{1}{|N_i|} \sum_{j \in N_i} h_j^{(l)}$$

What is the new corresponding transition matrix?

★ Solution ★

$$M = \frac{1}{2} I + \frac{1}{2} D^{-1} A \tag{3}$$

1.5 Over-Smoothing Effect (5 points)

In Question 1.1 we see that increasing depth could give more expressive power. On the other hand, however, a very large depth also gives rise to the undesirable effect of over smoothing. Assume we are still using the aggregation function from Question 1.3: $h_i^{(l+1)} = \frac{1}{|N_i|} \sum_{j \in N_i} h_j^{(l)}$. Show that the node embedding $h^{(l)}$ will converge as $l \to \infty$. Here we assume that the graph is connected and has no bipartite components. We also assume that the graph is undirected.

Over-smoothing thus refers to the problem of node embedding convergence. Namely, if all node embeddings converge to the same value then we can no longer distinguish them and our node embeddings become useless for downstream tasks. However, in practice, learnable weights, non-linearity, and other architecture choices can alleviate the over-smoothing effect.

Hint: Here are some properties that might be helpful:

- **A Markov Chain is *irreducible* if every state can be reached from every other state.**
- **A state i of a Markov Chain is *periodic* if, for a certain $t > 1$, it is only possible to travel from i to i in multiples of t timesteps. A Markov Chain is *aperiodic* if it has no periodic states.**
- **If a Markov Chain is irreducible and aperiodic, it will converge to a unique stationary distribution regardless of the starting state.**

You don't need to be super rigorous with your proof.

★ Solution ★

Aggregation can be written in matrix form.

$$h^{(l)} = M h^{(l-1)} = M^l h^{(0)} \tag{4}$$

where $M = D^{-1}A$ is the state transition matrix.
Use a normalized adjacency matrix $\tilde{A}$ that is similar to M (and therefore has the same set of eigenvalues):

$$\tilde{A} = D^{1/2}MD^{-1/2} = D^{1/2}(D^{-1}A)D^{-1/2} = D^{-1/2}AD^{-1/2} \tag{5}$$

Since $L_{sym} = I - D^{-1/2}AD^{-1/2} = I - \tilde{A}$, the number of connected components = the geometric multiple of the eigenvalue 0 of L_{sym} = the geometric multiple of the eigenvalue 1 of $\tilde{A} = 1$.
Furthermore, since M is a transition matrix, its spectral radius is $\rho(M) \leq 1$ *, and the graph is non-bisecting, so M also has no eigenvalue -1 *.
Any initial embedding vector $h^{(0)}$ can be expressed as a linear combination of the feature vectors u_i of matrix M:

$$h^{(0)} = c_1u_1 + c_2u_2 + ... + c_nu_n \tag{6}$$

$$h^{(l)} = M^l h^{(0)} = M^l(c_1u_1 + c_2u_2 + ... + c_nu_n) \tag{7}$$

$$= c_1(1)^l u_1 + c_2(\lambda_2)^l u_2 + ... + c_n(\lambda_n)^l u_n \tag{8}$$

$$\lim_{l\to\infty} h^{(l)} = c_1 \cdot 1 \cdot u_1 + 0 + ... + 0 = c_1\mathbf{1} \tag{9}$$

The vector $h^{(l)}$ converges to a vector where all elements are equal (c_1). This proves that after an infinite number of layers, the characteristic information of each node will be completely eliminated, leaving only a global average value (over-smoothing).

1.6 Learning BFS with GNN (7 points)

Next, we investigate the expressive power of GNN for learning simple graph algorithms. Consider breadth-first search (BFS), where at every step, nodes that are connected to already visited nodes become visited. Suppose that we use GNN to learn to execute the BFS algorithm. Suppose that the embeddings are 1-dimensional. Initially, all nodes have input feature 0, except a source node which has input feature 1. At every step, nodes reached by BFS have embedding 1, and nodes not reached by BFS have embedding 0. Describe a message function, an aggregation function, and an update rule for the GNN such that it learns the task perfectly.

★ Solution ★

- Message function:

$$m_j^{(l)} = h_j^{(l)} \tag{10}$$

- Aggregation function:

$$a_i^{(l)} = \sum_{j\in N_i} m_j^{(l)} \tag{11}$$

If $a_i^{(l)} > 0$, it means that at least one neighbor has been visited.

- Update rule:

$$h_i^{(l+1)} = \mathbb{1}(h_i^{(l)} + a_i^{(l)} \geq 1) \tag{12}$$

In this case, $\mathbb{1}(\cdot)$ is the indicator function, which returns 1 if the expression is true and 0 otherwise.

2 Node Embedding and its Relation to Matrix Factorization (24 points)

What to submit: For Q2.1, one or more sentences/equations describing the decoder. For Q2.2, write down the objective function. For Q2.3, describe the characteristics of W in one or more sentences. For Q2.4, write down the objective function. For Q2.5, characterize the embeddings, whether you think it will reflect structural similarity, and your justification. For Q2.6, one or more sentences for node2vec and struct2vec respectively. For Q2.7, one or more sentences of explanation. For Q2.8, one or more sentences characterizing embeddings from struct2vec.

Recall that matrix factorization and the encoder-decoder view of node embeddings are closely related. For the embeddings, when properly formulating the encoder-decoder and the objective function, we can find equivalent matrix factorization formulation approaches.

Note that in matrix factorization we are optimizing for L2 distance; in encoder-decoder examples such as DeepWalk and node2vec, we often use log-likelihood as in lecture slides. The goal to approximate A with $Z^T Z$ is the same, but for this question, stick with the L2 objective function.

2.1 Simple matrix factorization (3 points)

In the simple matrix factorization, the objective is to approximate adjacency matrix A by the product of embedding matrix with its transpose. The optimization objective is $\min_Z ||A - Z^T Z||_2$.

In the encoder-decoder perspective of node embeddings, what is the decoder? (Please provide a mathematical expression for the decoder)

★ Solution ★

The matrix A (original information of the graph) is being approximated by the product of the embedding matrix and its transpose matrix ($Z^T Z$). Each element at row i and column j of this product is calculated as:

$$(Z^T Z)_{ij} = z_i^T z_j \tag{13}$$

This is the inner product between the two embedding vectors of node i and node j.

2.2 Alternate matrix factorization (3 points)

In linear algebra, we define bilinear form as $z_i^T W z_j$, where W is a matrix. Suppose that we define the decoder as the bilinear form, what would be the objective function for the corresponding matrix factorization? (Assume that the W matrix is fixed)

★ Solution ★

$$\min_Z \|A - Z^T W Z\|_2 \tag{14}$$

2.3 BONUS: Relation to eigen-decomposition (3 points)

Recall eigen-decomposition of a matrix (link). What would be the condition of W, such that the matrix factorization in the previous question (2.2) is equivalent to learning the eigen-decomposition of matrix A?

★ Solution ★

For learning matrix decomposition to be equivalent to finding the eigenvalue decomposition of A, the required condition is that W must be a diagonal matrix. When W is a diagonal matrix, the elements on the main diagonal of W act as the eigenvalues (Λ) of matrix A. At this point, the rows of matrix Z will learn how to represent the corresponding eigenvectors.

2.4 Multi-hop node similarity (3 points)

Define node similarity with the multi-hop definition: 2 nodes are similar if they are connected by at least one path of length at most k, where k is a parameter (e.g. $k = 2$). Suppose that we use the same encoder (embedding lookup) and decoder (inner product) as before. What would be the corresponding matrix factorization problem we want to solve?

★ Solution ★

Based on the problem statement "maximum length is k", the matrix S will have elements defined as follows:

$$S_{ij} = \begin{cases} 1 & \text{if there is a path of length } 1, 2, \ldots, \text{ or } k \text{ between } i, j \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

$$S = \mathbb{1}\left(\sum_{m=1}^{k} A^m > 0\right) \tag{16}$$

2.5 node2vec & struct2vec (i) (3 points)

Finally, we'll explore some limitations of node2vec that are introduced in the lecture, and look at algorithms that try to overcome them.

As mentioned in the lecture, due to the way random walk works, it's hard for node2vec to learn structural embedding from the graph. Think about how a new algorithm called **struct2vec** works. For this question, we define a **clique** to be a fully connected graph, where any two nodes are connected.

Given a graph $G(V, E)$, it defines K functions $g_k(u, v)$, $k = 1, 2, .., K$, which measure the structural similarity between nodes. The parameter k means that only the local structures within distance k of the node are taken into account. With all the nodes in G, regardless of the existing edges, it forms a new clique graph where any two nodes are connected by an edge whose weight is equal to the structural similarity between them. Since struct2vec defines K structural similarity functions, each edge has a set of possible weights corresponding to $g_1, g_2, ..., g_K$.

The random walks are then performed on the clique. During each step, weights are assigned according to different g_k's selected by some rule (omitted here for simplification). Then, the algorithm chooses the next node with probability proportional to the edge weights.

Characterize the vector representations (i.e. the embedding space) of the 10-node cliques after running the **node2vec** algorithm on the graph in figure 2.1. Assume through the random walk, nodes that are close to each other have similar embeddings. Do you think the node embeddings will reflect the structural similarity? Justify your answer.

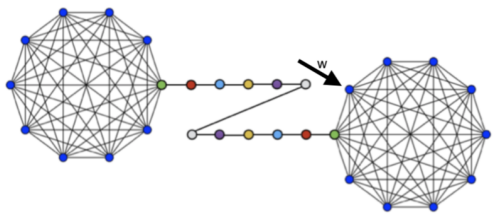


Figure 2.1: Two 10-node cliques

★ Solution ★

2.6 node2vec & struct2vec (ii) (3 points)

In the above figure 2.1, suppose you arrive at node w. What are the nodes that you can reach after taking one step further with the node2vec algorithm? What about with the struct2vec algorithm (suppose that for this graph, $g_k(u, v) > 0$ for any u,v,k)?

★ **Solution** ★

2.7 node2vec & struct2vec(iii) (3 points)

Why is it necessary to consider different g_k's during the random walk? (Please provide a more general answer, without limiting it to the specific example given)

★ **Solution** ★

2.8 node2vec & struct2vec (iv) (3 points)

Characterize the vector representations (i.e. the embedding space) of the two 10-node cliques after running the struct2vec algorithm on the graph in the above figure (Figure 2.1).

★ **Solution** ★

3 GCN (11 points)

Consider a graph $G = (V, E)$, with node features $x(v)$ for each $v \in V$. For each node $v \in V$, let $h_v^{(0)} = x(v)$ be the node's initial embedding. At each iteration k, the embeddings are updated as

$$h_{\mathcal{N}(v)}^{(k)} = \text{AGGREGATE}\left(\left\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v)\right\}\right)$$
$$h_v^{(k)} = \text{COMBINE}\left(h_v^{(k-1)}, h_{\mathcal{N}(v)}^{(k)}\right),$$

for some functions AGGREGATE$(\cdot)$ and COMBINE$(\cdot)$. Note that the argument to the AGGREGATE$(\cdot)$ function, $\left\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v)\right\}$, is a *multi-set*. That is, since multiple nodes can have the same embedding, the same element can occur in $\left\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v)\right\}$ multiple times. Finally, a graph itself may be embedded by computing some function applied to the multi-set of all the node embeddings at some final iteration K, which we denote as

$$\text{READOUT}\left(\left\{h_v^{(K)}, \forall v \in V\right\}\right)$$

We want to use the graph embeddings above to test whether two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic*. Recall that this is true if and only if

there is some bijection $\phi : V_1 \to V_2$ between nodes of G_1 and nodes of G_2 such that for any $u, v \in V_1$,

$$(u, v) \in E_1 \Leftrightarrow (\phi(u), \phi(v)) \in E_2$$

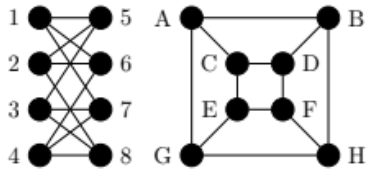
The way we use the model above to test isomorphism is as follows. For the two graphs, if their readout functions differ, that is

$$\text{READOUT}\left(\left\{h_v^{(K)}, \forall v \in V_1\right\}\right) \neq \text{READOUT}\left(\left\{h_v^{(K)}, \forall v \in V_2\right\}\right),$$

we conclude the graphs are *not* isomorphic. Otherwise, we conclude the graphs are isomorphic. Note that this algorithm is not perfect: graph isomorphism is thought to be hard! Below, we will explore the expressiveness of these graph embeddings.

3.1 Isomorphism Check (2 points)

Are the following two graphs isomorphic? If so, demonstrate an isomorphism between the sets of vertices. To demonstrate an isomorphism between two graphs, you need to find a 1-to-1 correspondence between their nodes and edges. If these two graphs are not isomorphic, prove it by finding a structure (node and/or edge) in one graph which is not present in the other.



★ **Solution** ★

3.2 Aggregation Choice (3 points)

The choice of the AGGREGATE$(\cdot)$ is important for the expressiveness of the model above. Three common choices are:

$$\text{AGGREGATE}_{\max}\left(\left\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v)\right\}\right)_i = \max_{u \in \mathcal{N}(v)}\left(h_u^{(k-1)}\right)_i \text{ (element-wise max)}$$

$$\text{AGGREGATE}_{\text{mean}}\left(\left\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v)\right\}\right) = \frac{1}{|\mathcal{N}(v)|}\sum_{u \in \mathcal{N}(v)}\left(h_u^{(k-1)}\right)$$

$$\text{AGGREGATE}_{\text{sum}}\left(\left\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v)\right\}\right) = \sum_{u \in \mathcal{N}(v)}\left(h_u^{(k-1)}\right)$$

Give an example of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and their initial node features, such that for some node $v_1 \in V_1$ and some node $v_2 \in V_2$ with the same initial features $h_{v_1}^{(0)} = h_{v_2}^{(0)}$, the updated features $h_{v_1}^{(1)}$ and $h_{v_2}^{(1)}$ are equal if we use mean and max aggregation, but different if we use sum aggregation.

Hint: Your node features can be scalars rather than vectors, i.e., one dimensional node features instead of n-dimensional. Also, you are free to arbitrarily choose the number of nodes and corresponding edges in your example.

★ **Solution** ★

3.3 Weisfeiler-Lehman Test (6 points)

Our isomorphism-test algorithm is known to be at most as powerful as the well-known *Weisfeiler-Lehman test* (WL test). At each iteration, this algorithm updates the representation of each node to be the set containing its previous representation and the previous representations of all its neighbors. The full algorithm is below.

Algorithm 3: Weisfeiler-Lehman Test

Data: $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, initial features $x(\cdot)$, number of iterations K
Result: Prediction of whether G_1 and G_2 are isomorphic
for $v \in V_1 \cup V_2$ **do**
  $l_v^{(0)} \leftarrow x(v)$
end
for $k = 1, \ldots, K$ **do**
  for $v \in V_1 \cup V_2$ **do**
    $l_v^{(k)} \leftarrow \text{HASH}\left(l_v^{(k-1)}, \{l_u^{(k-1)}, \forall u \in \mathcal{N}(v)\}\right)$
  end
end
return $\left\{l_v^{(K)}, \forall v \in V_1\right\} = \left\{l_v^{(K)}, \forall v \in V_2\right\}$

Prove that our neural model is at most as powerful as the WL test. More precisely, let G_1 and G_2 be non-isomorphic, and suppose that their node embeddings are updated over K iterations with the same AGGREGATE$(\cdot)$ and COMBINE$(\cdot)$ functions. Show that if

$$\text{READOUT}\left(\left\{h_v^{(K)}, \forall v \in V_1\right\}\right) \neq \text{READOUT}\left(\left\{h_v^{(K)}, \forall v \in V_2\right\}\right),$$

then the WL test also decides the graphs are not isomorphic.

Note: Your proof has to hold for any choice of AGGREGATE$(\cdot)$, COMBINE$(\cdot)$, and READOUT$(\cdot)$ functions. Namely, it's not sufficient to show this for a specific instance of the GNN model.

Hint: Try to proceed by contradiction: assume that the *Weisfeiler-Lehman* test *cannot* distinguish G_1 and G_2 after K iterations. Equivalently, after K iterations, READOUT$(\cdot)$ cannot distinguish the multi-sets of node embeddings of G_1 and G_2.

★ **Solution** ★