CS224W Homework 1

Siddhartha Parupudi

February 1, 2024

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

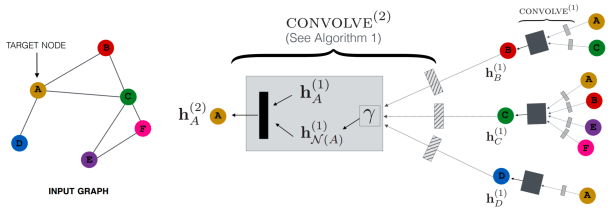


Figure 1.1: GNN architecture

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.

.

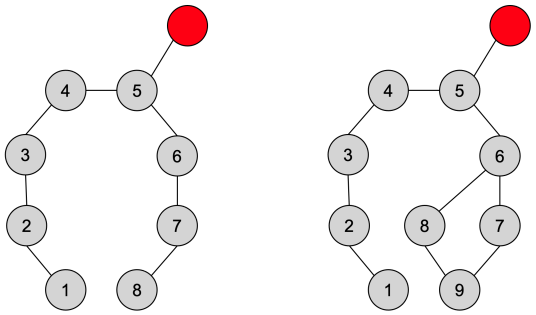


Figure 1.2: Figure for Question 1.1

Solution : We need 3 layers as it is in the 3rd layer that the neighboring nodes of the 2-hop neighbors start to differ, so for node 6, both 7,8 will be neighbors and their embeddings will be aggregated in the 2^{nd} graph, whereas only information from node 7 will be aggregated in the 1^{st} graph

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 row of M corresponds with the node ID in the graph?

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).

Solution : 1. The entires of the matrix M are $M_{ij} = \frac{1}{d_j}$ when there is an edge between node i and j. Therefore,

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

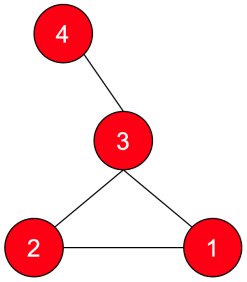


Figure 1.3: Figure for Question 1.2

2. The stationary distribution r is the eigenvector whose corresponding eigenvalue is 1 for the transition matrix M.

$$1 \cdot r = M \cdot r$$

Solving the above equations gives us that the distribution of the nodes is proportional to their degrees. Therefore,

$$r = \begin{bmatrix} 0.25 & 0.25 & 0.375 & 0.125 \end{bmatrix}^T$$

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 : The entires of the matrix M are $M_{ij} = \frac{1}{d_j}$ when there is an edge between node i and j. It is a column stochastic matrix. Each node shares it's importance equally to all of it's neighbors. So, we need $1/d_i$ terms. So we need D^{-1}. To make it column stochastic and share it's importance with it's neighbours, we multiply it with A. Therefore,

$$M = AD^{-1}$$

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 : Here, we include the embedding of the current node in the previous iteration as well. So we just add I to the previous expression with appropriate scaling.

$$M = \frac{1}{2}(I + AD^{-1})$$

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: Think about the Markov Convergence Theorem: Is the Markov chain irreducible and aperiodic? You don't need to be super rigorous with your proof.

Solution : Interpreting the graph as a markov chain with $D^{-1}A$ as the transition probability matrix. We see that the markov chain is **irreducible** as the graph is connected (we can get to every state from every other state) and it is **aperiodic** as well. Therefore, there exists a unique stationary distribution.

In the GNN, as we increase the number of layers, we aggregate information from the ℓ-hop neighbourhoods of the given node. If the number of layers is large, the ℓ-hop neighbourhoods of all the nodes will become same. So, when we are computing the messages, it is possible that the embeddings reach the stationary distribution of the markov chain and for all the subsequent layers, all the nodes will have the same embeddings and this will lead to the embeddings of all nodes converging to the same value.

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:** Just consider the 1-D embedding of each node
aggregate and update: Take $\max(\cdot)$ of the embeddings of all the neighbouring nodes.

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 decoder is simply the inner-product of the embeddings

$$\text{DEC}(z_u, z_v) = z_u^T z_v$$

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 : The new objective function is

$$\min_Z ||A - Z^T W Z||_2$$

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 : $A = Q\Lambda Q^T$. Therefore, the objective function is

$$\min_Z ||Q\Lambda Q^T - Z^T W Z||_2$$

. Therfore, if $W = \Lambda$, The learned embedding $Z \approx Q^T$ (the eigen-decomposition of A).

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 : The objective function is now

$$\min_Z ||A^k - Z^T Z||_2$$

.

The entries of A^k are positive if there exists a path of length k between the nodes

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.

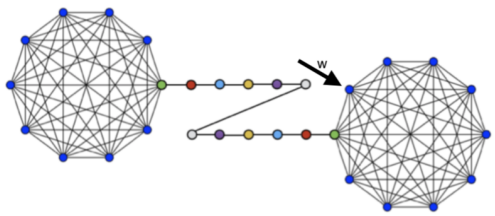


Figure 2.1: Two 10-node cliques

.

Solution : After running **node2vec**, the embeddings of the nodes of the cliques would be orthogonal to each other as it is highly improbable to go from one clique to the other as there is a long bridge in between them. While the embeddings of the nodes in the same clique will be highly similar as all the nodes are only 1-hop away from each other.

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 : We can reach all the nodes in that clique in 1 step.

2.7 node2vec & struct2vec (iii) (3 points)

Why is it necessary to consider different g_k's during the random walk?

Solution : If we don't consider different g_k's during the random walk, we are limited in our capability of characterising the node embeddings through sampling random walks and can't capture gloabal similarity features.

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 : If we choose a good choice of sampling from g_k's, after running **struct2vec**, we should expect that the embeddings of all the nodes in the cliques are very similar as they have identical structure.

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, $h_u^{(k-1)}, \forall u \in \mathcal{N}(v)$, is a *multi-set*. That is, since multiple nodes can have the same embedding, the same element can occur in $h_u^{(k-1)}, \forall u \in \mathcal{N}(v)$ 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 notate 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 $u, v \in V_1$,

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

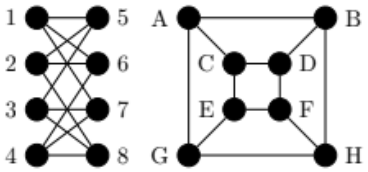
The way we use the model above to test isomorphism is the following. For the two graphs, if their readout functions differ, that is

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

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 : The above graph is isomorphic. The 1-1 mapping is :

$$
\begin{aligned}
1 &\to A \\
2 &\to H \\
3 &\to D \\
4 &\to E \\
5 &\to B \\
6 &\to G \\
7 &\to C \\
8 &\to F
\end{aligned}
$$

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:

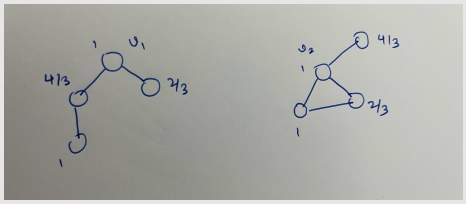
$$\text{AGGREGATE}_{\text{max}}\left(\left\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v)\right\}\right)_i = \max_{u \in \mathcal{N}(v)} \left(h_u^{(k-1)}\right)_i \text{ (element-wise max)}$$

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

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

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 (e.g. 3 nodes), their connections (i.e. edges between nodes) in your example.

Solution :



The above 2 graphs have same embeddings if we use **max** (4/3) and **mean** (1) aggregation but different **sum** (2,3) aggregations.

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: The proof has to be generic to any AGGREGATE, COMBINE, READOUT functions. Namely, it's not sufficient to show this for a specific instance of the GNN model.

Hint: You can use proof by contradiction by first assuming that *Weisfeiler-Lehman* test cannot decide whether G_1 and G_2 are isomorphic at the end of K'th iteration.

Solution : We first assume that WL test cannot decide between G_1 and G_2 in K iterations but that $readout(h_v^K, \forall v \in V_1) \neq readout(h_v^K, \forall v \in V_2)$.

Now, we prove that $\forall 0 \leq k < K$, after k iterations of WL test, both graphs have the same set of node labels and the same set of node neighborhoods, i.e.,

$$\{l_v^k, \forall v \in V_1\} = \{l_v^k, \forall v \in V_2\}$$

$$\{l_v^k, \{l_j^k : j \in N(v)\}, \forall v \in V_1\} = \{l_v^k, \{l_j^k : j \in N(v)\}, \forall v \in V_2\}.$$

Assume that the label set was not same then the WL test would have been able to decide that the graphs are non-isomorphic which is a contradiction. So, the label set is same for all k. Now assume that the nodes neighborhood set was not same then in the next iteration, the label set would have differed because different node neighborhood result in different labels so if node neighborhoods are not same then the labels in next iteration are too not same. This is again a contradiction so the node neighborhoods must also be same.

Now, we will consider the GNN iterations. If for any iteration k, $l_u^k = l_v^k$ where $u \in V_1$and $v \in V_2$, then $h_u^k = h_v^k$.

(i) Base case: For iteration 0, the statement trivially holds because we start with the same node features in both cases (starting with different features makes it useless to compare the two algorithms).

(il) Inductive Step: Let's say the statement holds for iteration n, we will now prove the statement for iteration n+1.

Now consider any u, v for which $l_u^{n+1} = l_v^{n+1}$, then,

$$\{l_u^k, \{l_j^k : j \in N(u)\}\} = \{l_v^k, \{l_j^k : j \in N(v)\}\}$$

Since our statement holds for the iteration n, we get

$$\{h_u^k, \{h_j^k : j \in N(u)\}\} = \{h_v^k, \{h_j^k : j \in N(v)\}\}$$

because if the labels or neighborhood are different then it means they had to be different in WL too which is a contradiction.
This now implies that $h_u^{n+1} = h_v^{n+1}$ because if the node label and node neighborhood are same, then using the same aggregate and combine function should

lead to the same output label. Hence, the induction statement holds for all iterations.

Now we take a look after the K^{th} iteration. We have assumed that WL test cannot decide between G_1 and G_2 in K iterations but that $readout(h_v^K, \forall v \in V_1) \neq readout(h_v^K, \forall v \in V_2)$. This means that the node labels set after the K iteration in WL test is same, i.e., $\{l_v^K, \forall v \in V_1\} = \{l_v^K, \forall v \in V_2\}$. But that means that $\{h_v^K, \forall v \in V_1\} = \{h_v^K, \forall v \in V_2\}$ which implies $readout(h_v^K, \forall v \in V_1) = readout(h_v^K, \forall v \in V_2)$. This is a contradiction. Hence proved that if WL can't decide then the GNN readouts will be same.