Assignment 4

Sadaf Fatollahy

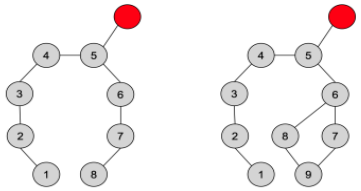
April 23, 2023

Professor : Dr.Taheri
Course name : Graph machine learning

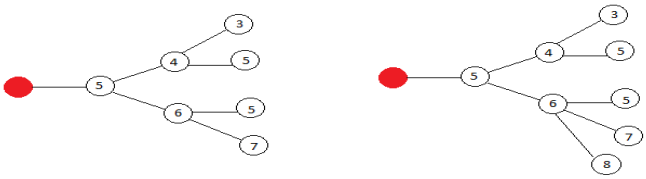
1 (2) GNN Expressiveness

1.1 (2.1) Effect of Depth on Expressiveness :

Consider the following 2 graphs, where all nodes have 1-dimensional initial feature vector$x = [1]$. We use a simplified version of GNN, with no nonlinearity,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 (Nv) and its current embeddingh_V^t to get$h_v^t + 1$. We run the GNNto 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 nodescan be distinguished (i.e., have different GNN embeddings)?



We must drow computation graph such as below .



According to the figure above 3 layers are enough.

1.2 (2.3) Relation to Random Walk :

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

Transition matrix. A random walk (or Markov chain), is most conveniently represented by its transition matrix P. P is a square matrix denoting the probability of transitioning from any vertex in the graph to any other vertex.

the steps to obtain the transition matrix T using the adjacency matrix A and degree matrix D are:

- Calculate the degree matrix D.
- Calculate the inverse of the degree matrix $D^{(-1)}$, taking care of vertices of degree 0.
- Multiply $D^{(-1)}$ by A to obtain the transition matrix T. The (i,j)-th element of T represents the probability of moving from vertex i to vertex j in one step of a random walk on the graph.

1.3 (2.6) Learning BFS with GNN :

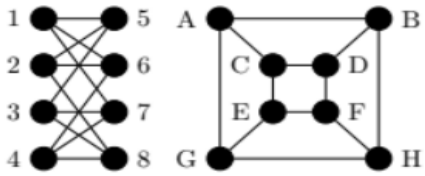
Next, we investigate the expressive power of GNN for learning simple graph algorithms. Consider breadth-first search (BFS), where at every step, nodes that are connected to already visited nodes become visited. Suppose that we use GNN to learn to execute the BFS algorithm. Suppose that the embeddings 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.

We consider massage function as identity. and aggregation function is Max. and update rule is a result of aggregation function.

2 (4) GCN

2.1 (4.1) Isomorphism Check:

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.



function :

A : 6 B : 2 C : 4 D : 8

E : 7 F : 3 G : 1 H : 5

2.2 (4.2) Aggregation Choice :

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

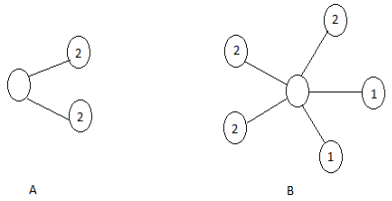
$$\text{AGGREGATE}_{\max}\left(\left\{h_u^{(k-1)}, \forall u \in \mathcal{N}(v)\right\}\right)_i = \max_{u\in\mathcal{N}(v)}\left(h_u^{(k-1)}\right)_i \text{ (element-wise max)}$$

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

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

Give an example of two graphs$G1 = (V1, E1)$ and $G2 = (V2, E2)$ and their initial node features, such that for two nodes $v_1 \in V_1$ and $v_2 \in V_2$ with the same initial features $h_{v_1}^{(0)} = h_{v_2}^{(0)}$, the updated features h_{v_1} and h_{v_2} 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.



for A :

$max = 2 \quad mean = 2 \quad sum = 4$

for B :

$max = 2 \quad mean = 2 \quad sum = 8$

2.3 (4.3) Weisfeiler-Lehman Test :

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 neighbours. The full algorithm is below. Prove that our neural model is at most as powerful as the

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 $\left\{l_v^{(K)}, \forall v \in V_1\right\} = \left\{l_v^{(K)}, \forall v \in V_2\right\}$

WL test. More precisely, let G1 and G2 be non-isomorphic, and suppose that their node embeddings are updated over K iterations with the same AGGREGATE(·) and COMBINE(·) 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. Hint: You can use proof by contradiction by first assuming that WeisfeilerLehman test cannot decide whether G1 and G2 are isomorphic at the end of K'th iteration.

Computaion graph is the same as unfolding tree , so we define :

Definition 3.3.1. The unfolding tree $\mathbf{T}_v^d$ of a node v up to depth d is

$$\mathbf{T}_v^d = \begin{cases} \text{Tree}(\ell_v) & \text{if } d = 0 \\ \text{Tree}(\ell_v, \mathbf{T}_{ne[v]}^{d-1}) & \text{if } d > 0 \end{cases}$$

where $\text{Tree}(\ell_v)$ is a tree constituted of a single node with label ℓ_v and $\text{Tree}(\ell_v, \mathbf{T}_{ne[v]}^{d-1})$ is the tree with the root node labeled with ℓ_v and having sub–trees $\mathbf{T}_{ne[v]}^{d-1}$. The set $\mathbf{T}_{ne[v]}^{d-1} = \{\mathbf{T}_{u_1}^{d-1}, \mathbf{T}_{u_2}^{d-1}, \dots\}$ collects all unfolding trees having depth $d-1$, with $u_i \in ne[v]$, $\forall i$.
Moreover, the *unfolding tree of* v, $\mathbf{T}_v = \lim_{d\to\infty} \mathbf{T}_v^d$, is obtained by merging all unfolding trees $\mathbf{T}_v^d$ for any d. ■

Two nodes u, v are said to be unfolding equivalent $u \sim_{ue} v$, if $T_u = T_v$. Analogously, two graphs G_1,G_2 are said to be unfolding equivalent$G_1 \sim_{ue} G_2$, if there exists a bijection between the nodes of the graphs that respects the partition induced by the unfolding equivalence on the nodes

The Weisfeiler–Lehman test algorithm :

1. At iteration 0, we set
$$c_v^{(0)} = \text{HASH}_0(\ell_v)$$
where HASH_0 is a function that bijectively codes every possible feature with a color in Σ.
2. For any iteration $t > 0$, we set
$$c_v^{(t)} = \text{HASH}((c_v^{(t-1)}, \{c_n^{(t-1)} | n \in ne[v]\}))$$
where HASH bijectively maps the above pair to a unique value in Σ, which has not been used in the previous iterations.

The algorithm terminates if the number of colors between two iterations does not change, i.e. when the cardinalities of $\{c_n^{(t-1)} | n \in V\}$ and $\{c_n^{(t)} | n \in V\}$ are equal.

Finally, we can introduce the equivalence that is induced on nodes by the WL test.

(WL–equivalence):

Two nodes, u and v, are said to be WL–equivalent, $u \sim_{WL} v$, if they have the same colour at the end of the 1–WL test, i.e. $c_u = c_v$. Analogously, two graphs, G_1 and G_2, are said to be WL–equivalent, $G_1 \sim_{WL} G_2$, if there exists a bijection between the coloring sets of G_1 and G_2 such that the corresponding sets have the same cardinality.

Theorem : Let $G = (V, E)$ be a labeled graph. Then, for each $u, v \in V$, $u \sim_{ue} v$ if and only if$u \sim_{WL} v$ holds.

Theorem : Let G_1 ,G_2 be two graphs. Then, $G_1 \sim_{ue} G_2$ if and only if $G_1 \sim_{WL} G_2$.

Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *be a graph and let* $u, v \in \mathbf{V}$, *with features* ℓ_u, ℓ_v. *Then,* $\forall t \in \mathbb{N}$

$$\mathbf{T}_u^t = \mathbf{T}_v^t \text{ iff } c_u^{(t)} = c_v^{(t)} \tag{4}$$

where $c_u^{(t)}$ *and* $c_v^{(t)}$ *represent the node coloring of* u *and* v *at time* t, *respectively.*

Proof. The proof is carried out by induction on t, which represents both the depth of the unfolding trees and the iteration step in the WL colouring.
For $t = 0$, $\mathbf{T}_u^0 = \text{Tree}(\ell_u) = \text{Tree}(\ell_v) = \mathbf{T}_v^0$ if and only if $\ell_u = \ell_v$ and $c_u^{(0)} = \text{HASH}_0(\ell_u) = \text{HASH}_0(\ell_v) = c_v^{(0)}$.
Let us suppose that Eq. (4) holds for $t - 1$, and prove that it holds also for t.

($\rightarrow$) Assuming that $\mathbf{T}_u^t = \mathbf{T}_v^t$, we have

$$\mathbf{T}_u^{t-1} = \mathbf{T}_v^{t-1} \tag{5}$$

and

$$\text{Tree}(\ell_u, \mathbf{T}_{ne[u]}^{t-1}) = \text{Tree}(\ell_v, \mathbf{T}_{ne[v]}^{t-1}) \tag{6}$$

By induction, Eq. (5) is true if and only if

$$c_u^{(t-1)} = c_v^{(t-1)} \tag{7}$$

Eq. (6) implies that $\ell_u = \ell_v$ and $\mathbf{T}_{ne[u]}^{t-1} = \mathbf{T}_{ne[v]}^{t-1}$, which means that an ordering on $ne[u]$ and $ne[v]$ exists s.t.

$$T_{ne_i(u)}^{t-1} = T_{ne_i(v)}^{t-1} \; \forall\, i = 1, \ldots, |ne[u]| \tag{8}$$

Hence, Eq. (8) holds iff an ordering on $ne[u]$ and $ne[v]$ exists s.t.

$$c_{ne(u)_i}^{t-1} = c_{ne(v)_i}^{t-1} \; \forall i = 1, \ldots, |ne[u]|$$

that is

$$\{c_m^{(t-1)} | m \in ne[u]\} = \{c_n^{(t-1)} | n \in ne[v]\} \tag{9}$$

Putting together Eqs. (7) and (9), we obtain:

$$\text{HASH}((c_u^{(t-1)}, \{c_m^{(t-1)} | m \in ne[u]\})) =$$
$$\text{HASH}((c_v^{(t-1)}, \{c_n^{(t-1)} | n \in ne[v]\}))$$

which implies that $c_u^{(t)} = c_v^{(t)}$

Let us assume that $c_u^{(t)} = c_v^{(t)}$; by definition,

$$\begin{aligned} &\text{HASH}((c_u^{(t-1)}, \{c_m^{(t-1)} | m \in ne[u]\})) = \\ &\text{HASH}((c_v^{(t-1)}, \{c_n^{(t-1)} | n \in ne[v]\})) \end{aligned} \tag{10}$$

Being the HASH function bijective, Eq. (10) implies that:

$$c_u^{(t-1)} = c_v^{(t-1)} \tag{11}$$

and

$$\{c_m^{(t-1)} | m \in ne[u]\} = \{c_n^{(t-1)} | n \in ne[v]\} \tag{12}$$

Eq. (11) is true if and only if, by induction,

$$\mathbf{T}_u^{t-1} = \mathbf{T}_v^{t-1} \tag{13}$$

which implies

$$\ell_u = \ell_v \tag{14}$$

Moreover, Eq. (12) means that an ordering on $ne[u]$ and $ne[v]$ exists such that

$$c_{ne(u)_i}^{t-1} = c_{ne(v)_i}^{t-1} \ \forall i = 1, \ldots, |ne[u]| \tag{15}$$

Instead, by induction, Eq. (15) holds iff an ordering on $ne[u]$ and $ne[v]$ exists so as $T_{ne_i(u)}^{t-1} = T_{ne_i(v)}^{t-1} \ \forall \ i = 1, \ldots, |ne[u]|$, i.e.

$$\mathbf{T}_{ne[u]}^{t-1} = \mathbf{T}_{ne[v]}^{t-1} \tag{16}$$

Finally, putting together Eqs. (14) and (16), we obtain

$$\text{Tree}(\ell_u, \mathbf{T}_{ne[u]}^{t-1}) = \text{Tree}(\ell_v, \mathbf{T}_{ne[v]}^{t-1})$$

that means $\mathbf{T}_u^t = \mathbf{T}_v^t$