CS224W Homework 1

February 1, 2023

1 Link Analysis (11 points)

REFRESHER

Recall from Colab 1 that PageRank measures importance of nodes in a graph using the link structure of the web. A "vote" from an important page is worth more. Specifically, if a page i with importance r_i has d_i out-links, then each link gets $\frac{r_i}{d_i}$ votes. Thus, the importance of a page j, represented as r_j is the sum of the votes on its in-links.

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

where d_i is the out degree of node i.

The PageRank algorithm (used by Google) outputs a probability distribution which represent the likelihood of a random surfer clicking on links will arrive at any particular page. At each time step, the random surfer has two options:

- Option 1: With prob. β, follow a link at random
- Option 2: With prob. $1 - \beta$, jump to a random page

Thus, the importance of a particular page is calculated with the following PageRank equation:

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1 - \beta)\frac{1}{N}$$

where N is the total number of nodes in the graph.

Personalizing PageRank is a very important real-world problem: different users find different pages relevant, so search engines can provide better results if they tailor their page relevance estimates to the users they are serving.

PageRank can be personalized with clever modifications of the teleport weights, which refer to the (possibly non-uniform) probabilities that a random surfer jumps to different pages if not following a link. The PageRank equation can be rewritten as:

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + \sum_{i} (1 - \beta) t_{i \to j} r_i$$

where $t_{i \to j}$ refers to the probability of jumping from page i to j under option 2. We have $\sum_k t_{i \to k} = 1$. Note that $t_{m \to k} = t_{n \to k} \forall m, n$. Here, $t_{m \to k}$ equals a constant probability for teleporting to node k. We call the set of all nodes k such that $t_{m \to k}$ is nonzero to be the teleport set.

You can refer to more information from the slides here.

For this question, the graph we're working on is the graph of webpages connected by hyperlinks, not bi-partite graphs.

Assume that people's interests are represented by a set of representative pages. For example, if Zuzanna is interested in sports and food, then we could represent her interests with the set of pages {`www.espn.com`, `www.epicurious.com`}. For notational convenience, we will use integers as names for webpages.

Suppose you have already computed the personalized PageRank vectors for the following users (the PageRank vector is the stationary distribution of a particular user over webpages):

1. User A, whose interests are represented by the teleport set {1, 2, 4}
2. User B, whose interests are represented by the teleport set {3, 4, 5}
3. User C, whose interests are represented by the teleport set {3, 5}
4. User D, whose interests are represented by the teleport set {2}

Assume that the weights for each node in a teleport set are uniform. Without looking at the graph (i.e. actually running the PageRank algorithm), can you compute the personalized PageRank vectors for the following users? If so, how? If not, why not? Assume a fixed teleport parameter β.

For the following questions (up to 1.3), assume that the PageRank vector for user i is v_i. Express your answer in terms of v_i if you can. For example, the PageRank vector for user A whose teleport set is {1, 2, 4} is v_A.

Hint: It may be helpful to write out the PageRank equations in matrix form.

1.1 Personalized PageRank I (2 points)

Eloise, whose interests are represented by the teleport set {1}.

1.2 Personalized PageRank I Cont'd (2 points)

Following 1.1, Felicity, whose interests are represented by the teleport set {5}.

1.3 Personalized PageRank I Cont'd (2 points)

Following 1.1, Glynnis, whose interests are represented by the teleport set {1, 2, 3, 4, 5} with weights

$$w = [0.1, 0.3, 0.2, 0.2, 0.2]$$

respectively.

1.4 Personalized PageRank II (5 points)

Suppose that you've already computed the personalized PageRank vectors of a set of users (denote the computed vectors V). What is the set of all personalized PageRank vectors that you can compute from V without accessing the web graph?

2 GNN Expressiveness (28 points)

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 4.1), 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.

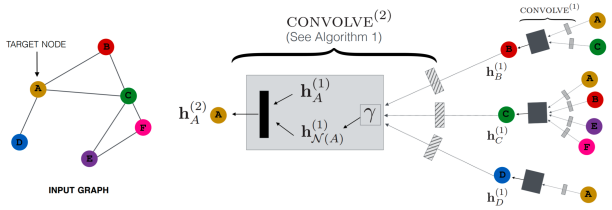


Figure 2.1: GNN architecture

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 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 Question 4.

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.

2.1 Effect of Depth on Expressiveness (4 points)

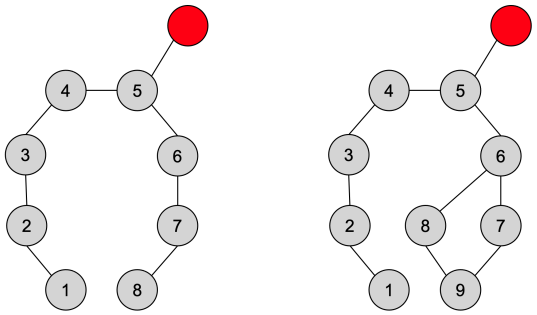
Consider the following 2 graphs, where all nodes have 1-dimensional initial feature vector $x = [1]$. We use a simplified version of GNN, with no nonlinearity, no learned linear transformation, and sum aggregation. Specifically, at every layer, the embedding of node v is updated as the sum over the embeddings of its neighbors (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)?

2.2 Random Walk Matrix (4 points)

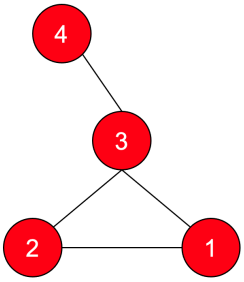
Consider the graph shown below.

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 rounded to the nearest



hundredth place and in the following form, e.g. $[1.00, 0.11, 0.46, 0.00]$. Note that before reporting you should normalize r (L2-norm).



2.3 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.

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

Suppose that we add a skip connection to the aggregation from Question 2.3:

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

What is the new corresponding transition matrix?

2.5 Over-Smoothing Effect (5 points)

In Question 2.1 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 2.3: $h_i^{(l+1)} = \frac{1}{|N_i|}\sum_{j\in N_i} h_j^{(l)}$. 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.

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.

2.6 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.

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

Recall in lecture 3 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.

3.1 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?

3.2 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?

3.3 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 3.2 is equivalent to learning the eigen-decomposition of matrix A?

3.4 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?

3.5 Limitation of node2vec (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 of the 10-node cliques after running the **node2vec** algorithm on the graph in the figure above. 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.

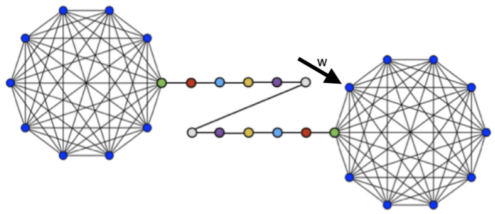


Figure 3.1: Two 10-node cliques

3.6 Limitation of node2vec (ii) (3 points)

In the above figure, 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)?

3.7 Limitation of node2vec (iii) (3 points)

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

3.8 Limitation of node2vec (iv) (3 points)

Characterize the vector representations of the two 10-node cliques after running the struct2vec algorithm on the graph in the above figure.

4 GCN (10 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 $u, v \in V_1$,

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

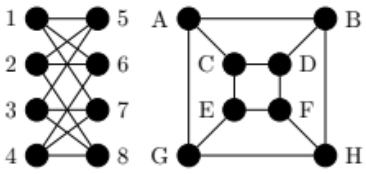
The way we use the model above to test isomorphism is the following. For the two graphs, if their readout functions differ, that is

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

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.

4.1 Isomorphism Check(1 point)

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.



4.2 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 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 $h_{v_1}^{(1)}$ 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.

4.3 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 neighbours. 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 em-

beddings 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.

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.