CS224W Homework 1

September 30, 2021

1 Link Analysis

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. Recall from class that PageRank can be specialized with clever modifications of the teleport vector. In this question, we will explore how this can be applied to personalize the PageRank algorithm.
For this question, the graph we're working on is the graph of webpages connected by hyperlinks as described in lectures, not the 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:

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

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, PageRank for user A whose teleport set is $\{1, 2, 3\}$ is v_A.

1.1 Personalized PageRank I

Eloise, whose interests are represented by the teleport set $\{2\}$.

1.2

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

1.3

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

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

respectively.

1.4 Personalized PageRank II

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?

1.5 A different equation for PageRank

In class you that the PageRank equation can be written in different ways. You will prove this in this question. Recall that the PageRank equation is:

$$\mathbf{r} = \mathbf{A}\mathbf{r}, \tag{1}$$

where $\mathbf{r} \in \mathbb{R}^N$ is the PageRank vector, i.e. the vector of PageRank scores for each of the N nodes, and (where $\mathbf{1}$ is the all-ones column vector of length N):

$$\mathbf{A} = \beta\mathbf{M} + \frac{1-\beta}{N}\mathbf{1}\mathbf{1}^\top. \tag{2}$$

Recall that here, $\mathbf{M}$ is the stochastic adjacency matrix, defined as

$$\mathbf{M}_{ij} = \left\{ \begin{array}{ll} \frac{1}{d_j}, & \text{if (j,i) is an edge} \\ 0 & \text{otherwise} \end{array} \right\} \tag{3}$$

where d_j is the degree of node j. Assume every node has degree at least 1.

Prove that (1) is equivalent to

$$\mathbf{r} = \beta\mathbf{M}\mathbf{r} + \frac{1-\beta}{N}\mathbf{1}. \tag{4}$$

Hint: Recall that $\mathbf{r}$ is normalized, i.e. $\sum_{i=1}^{N} \mathbf{r}_i = 1$.

2 Relational Classification I

As we discussed in class, we can use relational classification to predict node labels. Consider the graph G as shown in Figure 1 below. We would like to classify nodes into 2 classes "+" and "-". Labels for node 5, 6, 7 and 10 are given (blue for "+", red for "-"). Recall that using a probabilistic relational classifier to predict label Y_i for node i is defined as:

$$P(Y_i = c) = \frac{1}{|N_i|} \sum_{(i,j) \in E} W(i,j) P(Y_j = c)$$

where $|N_i|$ is the number of neighbors of node i. Assume all the edges have edge weight $W(i,j) = 1$ in this graph. For labeled nodes, initialize with the ground-truth Y labels, i.e., $P(Y_6 = +) = P(Y_7 = +) = P(Y_{10} = +) = 1.0, P(Y_5 = +) = 0$. For unlabeled nodes, use unbiased initialization $P(Y_i = +) = 0.5$. Update nodes by node ID in ascending order (i.e., update node 1 first, then node 2, etc.) After the second iteration, find

$$P(Y_i = +)$$

for i = 3,4,8

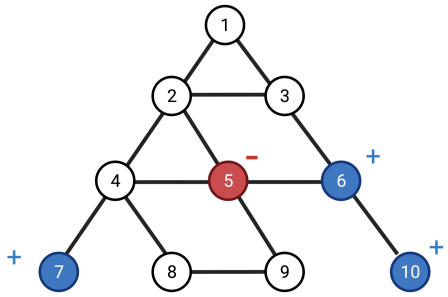


Figure 1: Graph G

2.1

$$P(Y_3 = +)$$

2.2

$$P(Y_4 = +)$$

2.3

$$P(Y_8 = +)$$

2.4

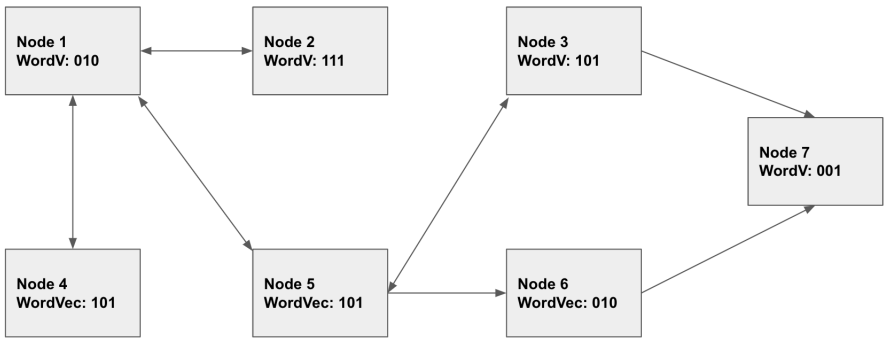
If we use 0.5 as the probability threshold, i.e., consider a node i belonging to class "+" if $P(Y_i = +) > 0.5$, which node will belong to class "-" after the second iteration? **(Your answer should be separated by comma, ordered by node ID, no extra space, e.g. 7,8,9 Include the final results with original nodes that are already assigned labels).**

3 Relational Classification II

Recall in iterative node classification, we classify a node i based on its attributes and the labels of its neighbor set. To do this, we utilize a feature vector a_i for each node i and use a local classifier to compute the best value for label Y_i in the iterative algorithm:

1. **Bootstrap phase:** Use local classifier f to compute initial label Y_i by setting Y_i to $f(a_i)$.
2. **Iteration phase:** Iterate until converge or max number of iterations: for each node i: 1) update node vector a_i, 2) update label Y_i to $f(a_i)$.

We will now apply iterative classification to the web page classification problem. We want to classify each node i to labels 0 or 1. Consider the graph consisting of the links between web pages below. The local features of each web page is a bag of words vector indicating the presence of words in a web page given by `WordV`. Each web page node also maintains a vector of neighborhood labels given by `LinkV`. This vector has the form (I_0, I_1) where $I_0 = 1$ if at least one of the incoming edges is coming from a page labeled 0 and $I_0 = 0$ otherwise, with similar definition for I_1.



We have two classifiers, f and g where f predicts label given `WordV` only, and g predicts label given `WordV` and `LinkV`. We will use f in the Bootstrap

phase and g in the Iteration phase. The results of f are given in Table 1. We describe the results of g with the piecewise equation:

$$g(\texttt{WordV}, \texttt{LinkV}) = \begin{cases} 0 & \text{if } I_0 = 1 \text{ or } \texttt{WordV} = 010 \\ 1 & \text{otherwise} \end{cases}$$

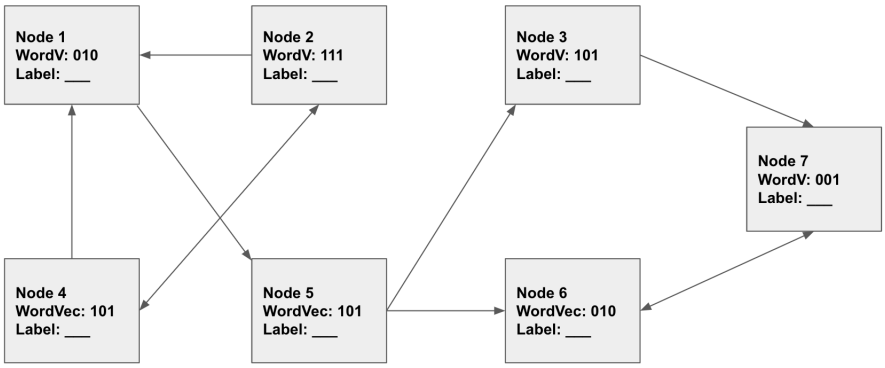
WordV	f(WordV)
001	0
010	0
101	1
111	1

Table 1: Results of classifier f on word-vector WordV.

For the remainder of this problem, we will run through two iterations of the Iterative Classifier on this web pages graph. (note that we're working on a **different** graph for the questions).

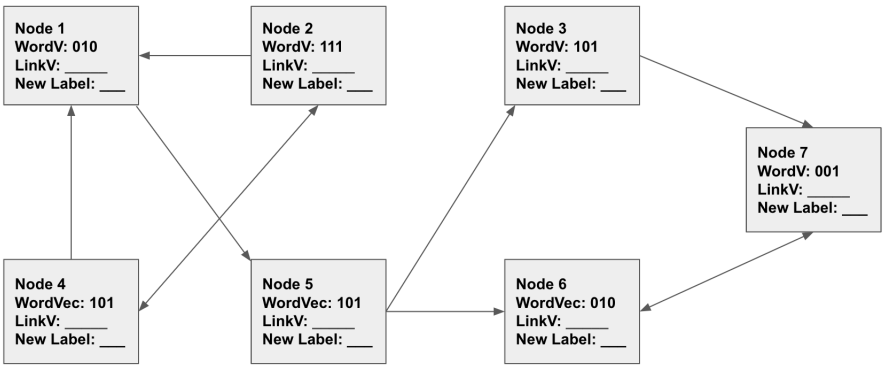
3.1 Bootstrap Phase

Use the trained word-vector classifier f to bootstrap our graph. Fill in the initial label for each node.



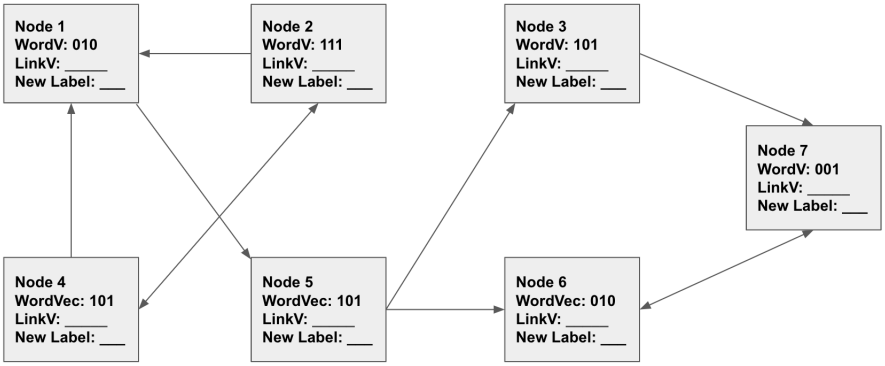
3.2 Iteration 1

First, using the initial labels from the bootstrap phase, compute the link vector LinkV for each node. Then, after computing the link vector for each node, compute the new label for each node.



3.3 Iteration 2

Repeat the iteration phase for one more iteration.



3.4 Convergence

Have the labels assigned by the iterative classification algorithm converged? If not, which nodes will change labels the next iteration?

4 GNN Expressiveness

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), 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 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:

$$h^{k+1} = \sigma\left(D^{-\frac{1}{2}}AD^{-\frac{1}{2}}h^k W^k\right) \tag{5}$$

where h^k represents the matrix of activations in the k-th layer, $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ 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
$z_v \leftarrow h_v^K, \forall v \in V$

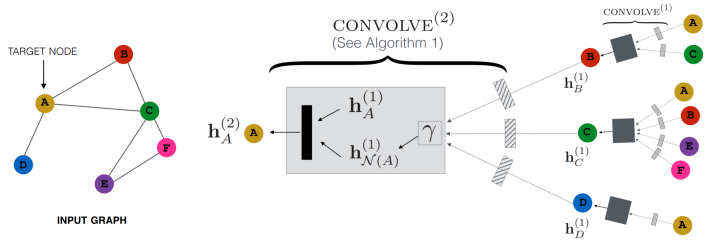
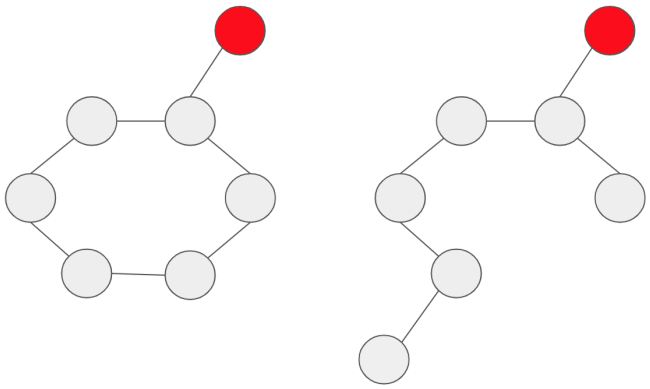


Figure 2: GNN architecture

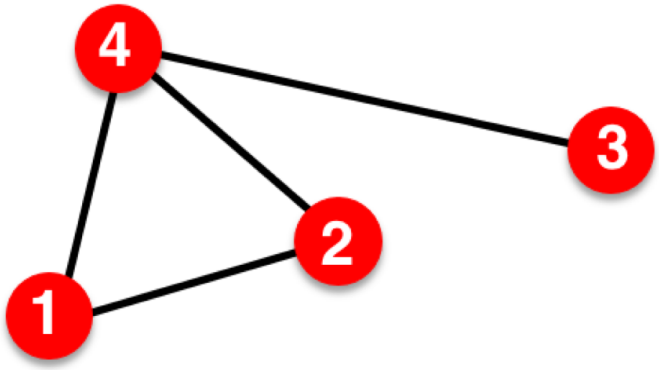
In this question, we investigate the effect of the number of message passing layers on the expressive power of Graph Convolutional Networks. In neural networks, expressiveness refers to the set of functions (usually the loss function for classification or regression tasks) a neural network is able to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

4.1 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 ($\mathcal{N}_v$) 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 4-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)?

4.2 Random Walk Matrix



i Assume that the current distribution is $r = \{0, 0, 1, 0\}$, and after the random walk, the distribution is $M * r$ (same as lecture). What is the

random walk transition matrix M, where each row of M corresponds with the node id in the graph.

ii 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).

4.3 Relation to Random Walk (i)

Let's explore the similarity between message passing and random walks. Let $h_i^{(l)}$ be the embedding of node i in layer l. Suppose that we are using a mean aggregator for message passing, and omit the learned linear transformation and non-linearity: $h_i^{(l+1)} = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} h_j^{(l)}$. 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 on 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.

4.4 Relation to Random Walk (ii)

Suppose that we add a skip connection in aggregation from Q4.3:

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

. What is the new corresponding transition matrix?

4.5 Over-Smoothing Effect

In Question 4.1 we see that increasing depth could give more expressive power. On the other hand, however, very large depth also gives rise to the undesirable effect of over smoothing. Assume we are still using the aggregation function from part 4.3 (i): $h_i^{(l+1)} = \frac{1}{|\mathcal{N}_i|} \sum_{|j \in \mathcal{N}_i|} h_j^{(l)}$. Show that the node embedding $h_i^{(l)}$ will converge as $\ell \to \infty$. 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 break down. However, in practice, learnable weights, non-linearity, and other architecture choices may alleviate the over-smoothing effect.

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

4.6 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.

5 Node Embedding and its relation to matrix factorization

Recall in lecture 4 that matrix factorization and the encoder-decoder view of node embeddings are closely related. When properly formulating the encoder-decoder and the objective function, we can find equivalent matrix factorization formulation for the embedding approaches.

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

5.1 Simple matrix factorization

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 $\texttt{min}_Z ||A - Z^T Z||_2$.

In the perspective of encoder-decoder perspective of node embeddings, what is the decoder?

5.2 Alternate matrix factorization

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

5.3 Relation to eigendecomposition

Recall eigenvector and eigenvalue definition in lecture 4. What would be the condition of W, such that the matrix factorization in 5.2 is equivalent to learning the eigen-decomposition of matrix A?

5.4 Multi-hop node similarity

Now we want to define node similarity using 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. 2). Suppose that we use the same encoder (embedding lookup) and decoder (inner product) as in question 5.2. What would be the corresponding matrix factorization problem we want to solve?

5.5 Limitation of node2vec (i)

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

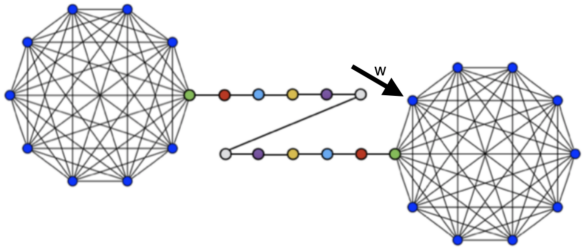
Remember it is mentioned in the lecture, that due to the way random walk works, it's hard for node2vec to learn structural embedding from the graph. Think about how a new algorithm called **struct2vec** works. For this question, we define **clique** to be fully connected graph, where any two nodes are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



5.6 Limitation of node2vec (ii)

In the above picture, suppose that 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)?

5.7 Limitation of node2vec (iii)

Why is there a need to consider different g_k's during the random walk?

5.8 Limitation of node2vec (iv)

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