1 Node Embedding (30 pts)

In this problem we explore two approaches to shallow embedding: LINE and Random Walk based embedding methods (DeepWalk). Recall that the central theme to define embedding vectors is to define a measurement of node similarity.

• In the LINE formulation, two nodes are similar if: (a) they are connected, also known as first-order proximity, or (b) their neighbors are similar, also known as second-order proximity.

• In the case of DeepWalk, two nodes are similar if they co-occur frequently in random walks. This random walk procedure is summarized as: given a starting point, randomly select a neighbor node, move to this neighbor, then repeat the process until the generated random path reaches a pre-determined length.

Suppose we want to perform node embedding for a graph $G = (V, E)$ using these two algorithms.

(a) For node embedding using LINE with first-order proximity, what's the time complexity of calculating all pairs of node similarities? How about second-order proximity? Suppose we now use a negative sampling scheme for $K << |V|$ negative samples, what's the time complexity in this case? (10 pts)

(b) Reason about why second-order proximity is required on top of first-order proximity. You may find the discussion in the original paper fruitful. (5 pts)

(c) Read the note from CS224W about Random walk-based approach and reason about comparison with LINE; in particular, the advantanges and disadvantages potentially associated with this approach. (10 pts).

(d) Reason (shortly) about the relation between Node2Vec and Word2Ve, from the perspective of sequence vs. graph data. (5 pts).

2 Knowledge Graph Embedding (15 pts)

In this problem, you are going to consider the TransE, DistMult, and RotatE models (slide09 P23-30).

You are going to answer the following questions and provide reasons.

(a) If we have a knowledge graph with friendship and enemy relationship, which model(s) of the TransE, DistMult, and RotatE can we use? Please explain your reason based on the score function of each model. (Hint: Friendship and enemy are symmetric relationships.) (5 pts)

(b) If we have a knowledge graph with father, grandfather, mother, and grandmother relationship, which model(s) can we use? Please explain your reason based on the score function. (Hint: The father of father is grandfather. The mother of mother is grandmother. Which model(s) can model composition relationship? How?) (5 pts)

(c) For each of TransE, DistMult, RotateE, provide an example (different from part (a) and (b)) for a scenario where it cannot model the particular relationship. (5 pts)

3 Bonus: Node Embedding and its Relation to Matrix Factorization (20 points)

For this bonus problem, if you want the extra credit, you need to do some research on your own. What to submit: For Q3.1, one or more sentences/equations describing the decoder. For Q3.2, write down the objective function. For Q3.3, describe the characteristics of W in one or more sentences. For Q3.4, write down the objective function. 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.

3.1 Simple matrix factorization (5 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)

3.2 Alternate matrix factorization (5 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)

3.3 BONUS: Relation to eigen-decomposition (5 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?

3.4 Multi-hop node similarity (5 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?

Answers:

1 Node Embedding (30 pts)

Q 1(a):

For node embedding using LINE with first-order proximity, the time complexity of calculating all pairs of node similarities is

$$O(|V|)$$

where $|V|$ is the number of nodes in the graph. This is because need to iterate through all nodes in the graph to calculate their similarities based on the first-order proximity.

For second-order proximity in LINE, the time complexity of calculating all pairs of node similarities depends on the average node degree in the graph. Let's denote the average node degree as d. In this case, the time complexity is approximately

$$O(d * |V|)$$

as that need to consider the neighbors of each node to calculate their similarities based on the second-order proximity.

Suppose a negative sampling scheme for $K << |V|$ negative samples in LINE. The time complexity remains

$$O(d * |V|)$$

because we still need to iterate through the neighbors of each node to calculate their similarities. The negative sampling scheme does not affect the time complexity in this case.

Q 1(b):

Second-order proximity is required on top of first-order proximity in the LINE algorithm to capture higher-order structural information and better reflect the node similarities in the graph. While first-order proximity considers direct connections between nodes, second-order proximity takes into account the similarity of their neighbors.

The inclusion of second-order proximity helps address the limitations of first-order proximity in capturing more complex relationships and structural patterns in the graph. By considering the similarities of neighboring nodes, the algorithm can capture the notion that nodes with similar neighbors are likely to have similar roles or functions in the graph.

Q 1(c):

The Random Walk-based approach offers advantages such as capturing global structure, scalability, unsupervised learning, and flexibility. However, it has potential drawbacks such as the loss of local information, sensitivity to parameters,

bias towards high-degree nodes, and the lack of edge information. The choice between LINE and DeepWalk depends on the specific characteristics of the graph and the desired trade-offs between local and global information capture.

There is a have summarization about the Random Walk-based approach:

Advantages of Random Walk-based approach (DeepWalk):

Capturing Global Structure: DeepWalk leverages random walks to explore the graph and capture global structural information. By generating random paths of nodes, DeepWalk can capture the overall connectivity patterns and relationships between nodes in the graph.

Scalability: DeepWalk is computationally efficient and scalable to large graphs. The random walk procedure can be parallelized and distributed, making it suitable for analyzing and embedding large-scale graphs.

Unsupervised Learning: DeepWalk is an unsupervised learning method, which means it does not require any labeled data or prior knowledge about the graph. It can automatically learn representations of nodes solely based on the graph structure.

Flexibility: DeepWalk is a flexible approach that can be applied to various types of graphs, including social networks, citation networks, and web graphs. It can capture both homogeneous and heterogeneous graphs, as long as a notion of node co-occurrence can be defined.

Disadvantages of Random Walk-based approach (DeepWalk):

Loss of Local Information: DeepWalk focuses on capturing global structural information through random walks. However, it may lose some local information or fine-grained details about individual nodes or specific local patterns in the graph. The generated random paths may not fully represent the intricate relationships within local neighborhoods.

Sensitivity to Parameters: DeepWalk involves setting parameters such as the random walk length and the number of random walks per node. The quality of the embeddings can be sensitive to these parameters, and finding the optimal values may require careful tuning.

Bias towards High-Degree Nodes: The random walk procedure in DeepWalk tends to spend more time in high-degree nodes since they have more neighbors to explore. As a result, the embeddings may be biased towards high-degree nodes, and the representation of low-degree nodes may be relatively weaker.

Lack of Edge Information: DeepWalk only considers the graph topology and does not take edge attributes or edge weights into account. It treats all edges equally during the random walk process, which may not fully capture the significance or importance of different edges in the graph.

Q 1(d):

Node2Vec adapts the skip-gram model of Word2Vec to learn embeddings from graph data by considering the local neighborhood structure of nodes, while Word2Vec operates on sequential data to learn embeddings based on the co-occurrence patterns of words. Both methods leverage the skip-gram model but operate on different types of data, sequence vs. graph, respectively.

2 Knowledge Graph Embedding (15 pts)

Q 2(a)

TransE and RotatE models are suitable for modeling friendship and enemy relationships in a knowledge graph due to their ability to capture symmetric relationships.

TransE uses a score function based on the L1 or L2 distance between the translated head entity and the tail entity, combined with the relation vector. Since friendship and enemy relationships are symmetric, the relation vector would be the same but in opposite directions. The L1 or L2 distance metric in TransE is insensitive to the direction of the relation vector, making it suitable for modeling symmetric relationships.

RotatE, on the other hand, is designed to handle complex-valued embeddings and uses rotation operations in the complex plane to model relations. The scoring function in RotatE calculates the plausibility of a triple based on the rotation angle between the embeddings of the head and tail entities. For friendship and enemy relationships, the rotation angle would be the same but with opposite signs. RotatE's ability to capture rotational patterns makes it well-suited for modeling symmetric relationships.

In contrast, the DistMult model may not be the best choice for modeling symmetric relationships. DistMult uses a simple element-wise multiplication as the scoring function, which lacks the flexibility to differentiate between opposite relations. As a result, DistMult would treat friendship and enemy relationships as identical because their embeddings would have the same magnitude and direction when multiplied element-wise.

Therefore, considering the symmetry of friendship and enemy relationships, both TransE and RotatE models are more appropriate choices compared to DistMult.

Q 2(b)

For a knowledge graph with father, grandfather, mother, and grandmother relationships, we can use the TransE and RotatE models. Both models have the ability to capture composition relationships, which are necessary to represent the hierarchical nature of father, grandfather, mother, and grandmother relationships.

TransE: TransE can model composition relationships by using the addition operation in its scoring function. For example, if we have the triples (father, x, y) and (father, y, z), TransE can infer the triple (grandfather, x, z) by adding the relation vectors of the two father relationships. Similarly, it can infer other composition relationships such as (mother, x, z) and (grandmother, x, z) by adding the corresponding relation vectors. TransE can capture these hierarchical dependencies through the addition operation in its scoring function.

RotatE: RotatE can also model composition relationships by utilizing the rotational patterns in the complex plane. For instance, if we have the triples

(father, x, y) and (father, y, z), RotatE can infer the triple (grandfather, x, z) by rotating the relation vector of the father relationship. Similarly, it can infer other composition relationships such as (mother, x, z) and (grandmother, x, z) by applying the appropriate rotations to the relation vectors. RotatE's ability to perform rotations in the complex plane allows it to capture composition relationships in the knowledge graph.

Both TransE and RotatE can effectively model the composition relationships in the given knowledge graph, enabling the inference of higher-level relationships such as grandfather, grandmother, and so on

Q 2(c)

Here are examples

TransE: TransE may struggle to model relationships that involve many-to-many mappings or complex patterns. For example, consider a knowledge graph with a "siblings" relationship. TransE represents relations as vector translations, but it may have difficulty capturing the complexity of sibling relationships, where multiple entities can have a shared sibling. The model's simple translation-based approach may not be able to capture the nuances of such relationships.

DistMult: DistMult may face challenges in modeling relationships that require more expressive interactions between entities and relations. For instance, consider a knowledge graph with a "works-with" relationship that represents collaboration between individuals. DistMult's scoring function, which involves element-wise multiplication, assumes a simple and symmetric interaction pattern. However, the "works-with" relationship may involve more complex and context-dependent interactions that DistMult may not be able to fully capture.

RotatE: RotatE may struggle to model relationships that are inherently asymmetric or anti-symmetric. For example, consider a knowledge graph with an "opposes" relationship that represents opposing viewpoints or ideologies. RotatE's rotational patterns in the complex plane are based on symmetry, which may not align well with the asymmetry or anti-symmetry present in the "opposes" relationship. The model's reliance on complex rotations may not be suitable for accurately representing such relationships.

3 Bonus: Node Embedding and its Relation to Matrix Factorization (20 points)

Q3.1

In the encoder-decoder perspective of node embeddings, the decoder is responsible for reconstructing the adjacency matrix based on the learned node embeddings. In the context of matrix factorization, the decoder corresponds to the reconstructed adjacency matrix, which can be expressed as follows:

Let's assume we have N nodes and K-dimensional node embeddings. The decoder reconstructs the adjacency matrix A_hat using the learned node embeddings Z as:

$$\hat{A} = Z * Z^T$$

Here, Z is an NxK matrix representing the node embeddings, and $\hat{A}$ is the reconstructed adjacency matrix. The matrix multiplication $Z * Z^T$ produces an NxN matrix, where the entry $\hat{A}[i][j]$ represents the reconstructed edge weight or similarity between nodes i and j.

The goal of the optimization objective, as mentioned earlier, is to minimize the squared L2 distance between the original adjacency matrix A and the reconstructed adjacency matrix $\hat{A}$, which can be written as:

$$minimize||A - \hat{A}||^2$$

In the context of matrix factorization, this optimization objective aims to find the optimal node embeddings Z that can best reconstruct the adjacency matrix A.

Q3.2

If the decoder is defined as the bilinear form $z_i^T W z_j$, where W is a fixed matrix, the objective function for the corresponding matrix factorization would be to minimize the squared L2 distance between the original adjacency matrix A and the reconstructed adjacency matrix $\hat{A}$.

The objective function can be written as:

$$minimize||A - \hat{A}||^2,$$

where A is the original adjacency matrix and $\hat{A}$ is the reconstructed adjacency matrix using the bilinear form decoder.

To express this objective function in terms of the node embeddings Z, we need to substitute the decoder expression into the objective function. Using the bilinear form decoder, the reconstructed adjacency matrix $\hat{A}$ can be written as:

$$\hat{A} = Z * W * Z^T.$$

Substituting this into the objective function, we have:

$$minimize||A - Z * W * Z^T||^2.$$

Here, Z is an NxK matrix representing the node embeddings, and W is a fixed matrix.

Therefore, the objective function for the matrix factorization using the bilinear form decoder is to minimize the squared L2 distance between the original adjacency matrix A and the reconstructed adjacency matrix $\hat{A}$, given by $||A - Z * W * Z^T||^2$. The optimization is performed over the node embeddings matrix Z. The goal is to find the optimal node embeddings that can best reconstruct the adjacency matrix A using the bilinear form decoder.

Q3.3

The matrix factorization in the previous question 3.2 would be equivalent to learning the eigen-decomposition of matrix A if the matrix W is a diagonal matrix with the eigenvalues of A as its diagonal elements.

Specifically, for the matrix factorization to be equivalent to learning the eigen-decomposition of A, the following condition should hold for W:

W should be a diagonal matrix: Each off-diagonal element of W is zero, and only the diagonal elements are non-zero.

The diagonal elements of W should correspond to the eigenvalues of A: The i-th diagonal element of W should be equal to the i-th eigenvalue of A.

By satisfying these conditions, the bilinear form decoder in the matrix factorization captures the eigen-decomposition of A, where the node embeddings Z play the role of the eigenvectors, and the diagonal matrix W represents the eigenvalues. The reconstructed adjacency matrix $\hat{A}$ can be seen as an approximation of A using the eigen-decomposition.

While this condition makes the matrix factorization equivalent to learning the eigen-decomposition, it is not necessary for general matrix factorization. Matrix factorization techniques can be used to learn latent representations without explicitly modeling the eigen-decomposition of the matrix.

Q3.4

In the context of multi-hop node similarity, the corresponding matrix factorization problem we want to solve is to learn node embeddings that capture the underlying structure of the graph, with the objective of maximizing the similarity between connected nodes within a certain hop distance.

To formalize this, let's denote the adjacency matrix of the graph as A, where $A[i][j]$ represents the edge weight or connectivity between nodes i and j. The objective is to find node embeddings Z that can reconstruct the adjacency matrix A, emphasizing the similarity between nodes connected by paths of length at most k.

The corresponding objective function for this matrix factorization problem can be defined as follows:

$$minimize||A - Z * Z^T||^2,$$

where Z is an NxK matrix representing the node embeddings, N is the number of nodes in the graph, and K is the dimensionality of the embeddings.

In this objective function, we aim to minimize the squared L2 distance between the original adjacency matrix A and the reconstructed adjacency matrix $\hat{A}$, where $\hat{A}$ is given by the inner product of the node embeddings Z:

$$\hat{A} = Z * Z^T$$

. By optimizing this objective function, the learned node embeddings Z will capture the inherent structure of the graph, emphasizing the similarity between nodes connected by paths of length at most k. The matrix factorization problem aims to find the optimal node embeddings that best reconstruct the adjacency matrix, with an emphasis on preserving the multi-hop node similarity.