CS224W Homework 3

Due: November 14, 2024

1 GNNs as MLP of eigenvectors [20 points]

1.1 Batch Node Update [2 points]

Consider the update for Graph Isomorphism Network:

$$\mathbf{x}_v^{(l+1)} = \texttt{MLP}\left((1+\epsilon)\,\mathbf{x}_v^{(l)} + \sum_{u\in\mathcal{N}(v)} \mathbf{x}_u^{(l)}\right), \tag{1}$$

where $\mathbf{x}_v^{(l)} \in \mathbb{R}^{d_l}$ is the embedding of node v at layer l. Let $\mathbf{X}^{(l)} \in \mathbb{R}^{N\times d_l}$ be a matrix containing the embeddings of all the nodes in the graph, i.e., $\mathbf{X}^{(l)}[:, v] = \mathbf{x}_v^{(l)}$. Also, let $\mathbf{A} \in \{0,1\}^{N\times N}$ represent the adjacency matrix of the graph. Write down the update of $\mathbf{X}^{(l+1)}$ as a function of $\mathbf{X}^{(l)}$ and $\mathbf{A}$.

★ Solution ★

1.2 Single Layer MLP [2 points]

Assume that $\texttt{MLP}()$ represents a single layer MLP with no bias term. Write down the update of $\mathbf{X}^{(l+1)}$ as a function of $\mathbf{X}^{(l)}$ and $\mathbf{A}$, and the trainable parameters $\mathbf{W}^{(l)}$ of layer l.

★ Solution ★

1.3 Eigenvector Extension [4 points]

Let $\{\lambda_n, \mathbf{v}_n\}_{n=1}^N$ represent the eigenvalues and eigenvectors of the graph adjacency. Then we can write $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$, where $\mathbf{V} \in \mathbb{R}^{N\times N}$ is the matrix of eigenvectors with $\mathbf{V}[:, n] = \mathbf{v}_n$ and $\mathbf{\Lambda} \in \mathbb{R}^{N\times N}$ is the diagonal matrix of eigenvalues with $\mathbf{\Lambda}[n, n] = \lambda_n$. Show that

$$\mathbf{X}^{(l+1)} = \sigma\left(\mathbf{V}\hat{\mathbf{W}}^{(l)}\right), \quad \hat{\mathbf{W}}^{(l)}[n, j] = (\lambda_n + 1 + \epsilon)\sum_{i=1}^{d_l} \mathbf{W}^{(l)}[i, j]\langle \mathbf{v}_n, \mathbf{X}^{(l)}[:, i]\rangle,$$

where $\langle \cdot \rangle$ denotes the dot product. Hint: Use the fact that the eigenvectors are orthonormal. Next, show that each feature across all nodes, $\mathbf{X}^{(l+1)}[:, i]$, can be expressed as a linear combination of eigenvectors, followed by a pointwise nonlinearity.

★ Solution ★

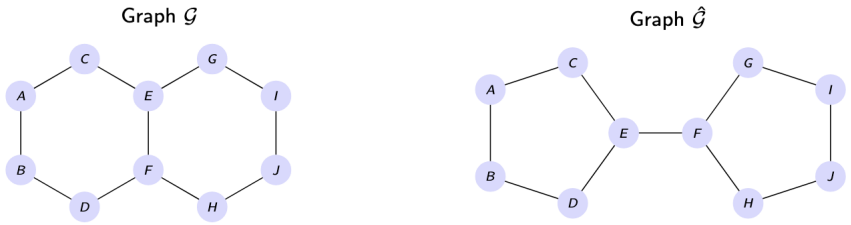
1.4 GraphSAGE [4 points]

Perform the same analysis for the GraphSAGE update when the aggregation function is sum pooling. Recall that the GraphSAGE update function is

$$\begin{aligned}\mathbf{x}_v^{(l+1)} &= \sigma\left(\mathbf{W}^{(l)} \cdot \text{CONCAT}\left(\mathbf{x}_v^{(l)}, \mathbf{x}_{N(v)}^{(l)}\right)\right) \\ &= \sigma\left(\mathbf{W}_1^{(l)}\mathbf{x}_v^{(l)} + \mathbf{W}_2^{(l)}\text{AGG}\left(\mathbf{x}_u^{(l)}, \forall u \in N(v)\right)\right)\end{aligned}$$

★ Solution ★

1.5 Eigendecomposition Analysis [8 points]



For graphs $\mathcal{G}$ and $\hat{\mathcal{G}}$ instantiate the graph adjacencies in Numpy, PyTorch, or PyG, and compute their eigenvalue decompositions. What do you observe?

★ Solution ★

Consider a GIN where all nodes start with the same initial color, i.e., $\mathbf{x}_v^{(0)} = 1$ for all nodes $v \in \mathcal{V}$. This setup is equivalent to having $\mathbf{X}^{(0)} = \mathbf{1}$, where $\mathbf{1}$ denotes the all-one vector. This is the initialization of the WL test. Using the equations in 1.3, derive the expression for $\mathbf{X}^{(1)}$.

★ Solution ★

Observe that each column $\mathbf{X}^{(1)}[:, j]$ is a linear combination of eigenvectors, followed by a pointwise nonlinearity. What is the weight associated with each eigenvector? What factors determine this weight?

★ Solution ★

Compute the dot product $\langle \mathbf{v}_n, \mathbf{X}^{(0)} \rangle$, for each eigenvector across both graphs. What do you observe?

★ Solution ★

What does the previous result suggest about $\mathbf{X}^{(1)}$ for the graphs $\mathcal{G}$ and $\hat{\mathcal{G}}$?

★ Solution ★

2 LightGCN [25 points]

We learned in class about **LightGCN**, a GNN model for recommender systems. Given a bipartite user-item graph $G = (V, E)$, let $\mathbf{A} \in \mathbb{R}^{|V|\times|V|}$ be its unnormalized adjacency matrix, $\mathbf{D} \in \mathbb{R}^{|V|\times|V|}$ be its degree matrix and $\mathbf{E}^{(k)} \in \mathbb{R}^{|V|\times d}$ be its node embedding matrix at layer k where d is the embedding dimension. Let $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ be the normalized adjacency matrix.

The original GCN updates node embeddings across layers according to $\mathbf{E}^{(k+1)} = \text{ReLU}(\tilde{\mathbf{A}}\mathbf{E}^{(k)}\mathbf{W}^{(k)})$, while LightGCN removes the non-linearity and uses the equation for each layer $k \in \{0, 1, ..., K-1\}$:

$$\mathbf{E}^{(k+1)} = \tilde{\mathbf{A}}\mathbf{E}^{(k)} \tag{2}$$

Moreover, LightGCN adopts multi-scale diffusion to compute the final node embeddings for link prediction, averaging across layers:

$$\mathbf{E} = \sum_{i=0}^{K} \alpha_i \mathbf{E}^{(i)}, \tag{3}$$

where we have uniform coefficients $\alpha_i = \frac{1}{K+1}$.

2.1 Advantages of Average Embeddings [4 points]

Why does LightGCN average over layer embeddings? What benefits does it bring, in a recommendation systems setting?

What to submit? 1-3 sentences of explanation on the reasons and benefits of averaging across layers.

★ Solution ★

2.2 Self-connection [4 points]

We denote the embedding of an item i at layer-k $\mathbf{e}_i^{(k)}$ and that of a user u $\mathbf{e}_u^{(k)}$. The graph convolution operation (a.k.a., propagation rule) in LightGCN is defined as:

$$\mathbf{e}_u^{(k+1)} = \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i^{(k)}$$

$$\mathbf{e}_i^{(k+1)} = \sum_{u \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|}} \mathbf{e}_u^{(k)}$$

The symmetric normalization term $\frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}}$follows the design of standard GCN, which can avoid the scale of embeddings increasing with graph convolution operations.

However, from the equations above, we can find that in LGCN, we only aggregate the connected neighbors and do not integrate the target node itself (i.e., there is no **self-connection**). This is different from most existing graph convolution operations that typically aggregate extended neighbors and also specifically handle self-connection.

Does LightGCN contain implicit self-connection? If your answer is yes, which operation captures the same effect as self-connection? If no, what do you think is the reason why LightGCN doesn't need self-connection or similar effects?

What to submit? Yes or no and 1-2 sentences of justification.

★ Solution ★

2.3 Relation with APPNP [5 points]

There is a work that connects GCN with Personalized PageRank, where the authors propose a GCN variant named APPNP that can propagate long range without the risk of oversmoothing. Inspired by the teleport design in Personalized PageRank, APPNP complements each propagation layer with the starting features (i.e., the 0-th layer embeddings), which can balance the need of preserving locality (i.e., staying close to the root node to alleviate oversmoothing) and leveraging the information from a large neighborhood. The propagation layer in APPNP is defined as:

$$\mathbf{E}^{(k+1)} = \beta \mathbf{E}^{(0)} + (1-\beta)\tilde{\mathbf{A}} E^{(k)}$$

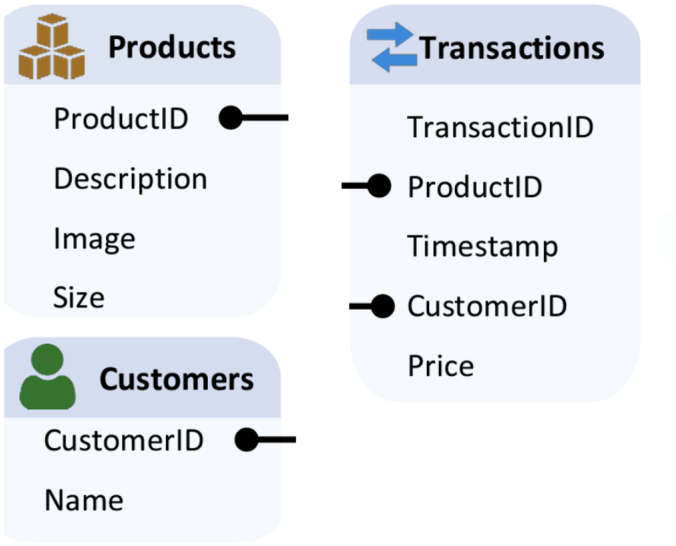
where β is called the "teleport probability" to control the retention of starting features in the propagation, and $\tilde{\mathbf{A}}$ denotes the normalized adjacency matrix.

Aligning with Equation (3), we can see that by setting α_k accordingly, LightGCN can fully recover the prediction embedding used by APPNP. As such, LightGCN shares the strength of APPNP in combating oversmoothing — by setting the α properly, LightGCN allows using a large K for long-range modeling with controllable oversmoothing.
Express the layer-K embeddings $\mathbf{E}^{(K)}$ of APPNP as a function of the initial embeddings $\mathbf{E}^{(0)}$ and the normalized adjacency matrix $\tilde{\mathbf{A}}$. Show all work.

What to submit? Multi-line mathematical derivation of the relationship between $\mathbf{E}^{(K)}$ and $\mathbf{E}^{(0)}$

★ Solution ★

3 Relational Deep Learning [15 points]



Assume we have the relational database as seen above, which consists of three tables. These tables contain information about products, customers, and transactions in which customers purchase products. Each table contains a unique identifier, known as a *primary key*, potentially along with other attributes. *Foreign keys* in a table create connections between tables by referencing primary keys in other tables. In the three tables shown above, "ProductID", "TransactionID", and "CustomerID" are the primary keys, while "ProductID" and "CustomerID" are also foreign keys for the "Transactions" table.

3.1 Schema Graphs [1 point]

A key component of a relational deep learning framework is the schema graph, which illustrates the relationships between tables in a database. In a schema graph, each table is represented as a node, and an edge is drawn between two nodes if a primary key from one table appears as a foreign key in another. This graph helps visualize how data is linked across the database.

Describe or draw what the schema graph of this database would look like (hint: it's very simple).

★ Solution ★

3.2 Relational Entity Graphs [4 points]

Table 1: Products

ProductID	Description	Image	Size
1	Smartphone	[]	Small
2	Laptop	[]	Medium
3	TV	[]	Large
4	Headphones	[]	Small

Table 2: Customers

CustomerID	Name
101	Alice
102	Bob
103	Carol

Table 3: Transactions

TransactionID	ProductID	Timestamp	CustomerID	Price ($)
1001	1	2024-10-15	101	600
1002	3	2024-10-20	102	500
1003	2	2024-10-26	103	1,300
1004	4	2024-11-01	101	100
1005	1	2024-11-02	101	600
1006	2	2024-11-12	103	1,300

Another component of this framework is the relational entity graph. The nodes of this graph are all the individual entities rather than tables. Links are again made by primary-foreign key connections - that is, two entities are linked if they appear together in the same entry of any table in the database. Given the list of transactions above, produce a relational entity graph describing this database.

★ Solution ★

3.3 Computation Graphs [6 points]

The computational graphs used for training are dependent on the specific timestep used for prediction. For example, let's assume our training table (which defines the information we seek to predict) contains the following information:

a. Target: How much total money a customer spends in the next 30 days

b. ID: Customer ID

c. Timestep: The time at which the 30 day period starts

When predicting, we can only use the information in the database that takes place before our prediction period. That means the computational graphs (the specific set of nodes and connections we send messages over) we use for predictions are directly dependent on the timestep in our training table. Let's say we want to make predictions for customer 101. Using the tables from the previous part, draw out the computation graphs if we wanted to make predictions at 2024-10-20, 2024-11-01, 2024-11-12.

★ Solution ★

3.4 Message Passing [4 points]

A relational database will produce a heterogeneous graph. What are example message passing and update rules that can be used to make predictions like the one mentioned above?

★ Solution ★

4 Honor Code [0 points]

(X) I have read and understood Stanford Honor Code before I submitted my work.

Collaboration: Write down the names & SUNetIDs of students you collaborated with on Homework 3 (None if you didn't).

Note: Read our website on our policy about collaboration!