# CS224W Homework 3

Due: November 13, 2025

## 1 Graph Transformers [20 points]

*Coverage: This problem is covered in Lecture 8.*

We study how standard Transformer components (tokens, positional encodings, and self-attention) can be adapted to graphs, how attention relates to message passing, and how to inject structure and edge features into the attention mechanism.

### 1.1 Self-Attention as Message Passing [4 points]

Recall the single-head self-attention update with inputs $X \in \mathbb{R}^{n\times d}$:

$$\mathrm{Att}(X) = \mathrm{softmax}\big(QK^\top\big)V, \quad Q = XW_Q,\ K = XW_K,\ V = XW_V.$$

Focusing on node $i$, one can write

$$\mathbf{z}_i = \sum_{j=1}^{n} \underbrace{\mathrm{softmax}_j\left(\mathbf{q}_i^\top \mathbf{k}_j\right)}_{\alpha_{ij}} \mathbf{v}_j.$$

Show that if we *mask* the softmax to only sum over neighbors $j \in \mathcal{N}(i)$ (and optionally $i$ itself), namely we set $\alpha_{ij} = 0$ for non-neighbors by applying an adjacency mask, the update reduces to a message passing GNN layer akin to GAT. Write the masked update and explain the correspondence between messages and aggregation.

**What to submit?** The masked equation and 2–3 sentences explaining why this is message passing.

**★ Solution ★**

### 1.2 Designing Tokens and Positional Encodings [8 points]

A graph Transformer must decide (i) what the *tokens* are, and (ii) how to inject *position/structure*. Consider node-level prediction on a graph $G$ with node features $X$.

(a) Suppose we design a graph Transformer that treats each **node** as a token and uses **Laplacian eigenvectors** for positional encodings (PEs).

Let $P \in \mathbb{R}^{n\times k}$ be the PE matrix derived from the graph Laplacian. (i) Clearly describe how $P$ is constructed. (ii) If we do not simply add $P$ to the node feature matrix $X$, explain how $P$ and $X$ are combined to form the Transformer's input representation for each node.

**What to submit:** A precise definition of $P$ and a short formula or description showing how $X$ and $P$ are combined into the model input.

(b) Briefly state one strength and one limitation of Laplacian-eigenvector PEs.

**What to submit?** 2 sentences describing the strength and limitation repectively.

(c) Explain in 1–2 sentences why using the P you constructed above as PEs can lead to the drawback you mentioned.

**What to submit?** 1-2 sentences.

(d) Propose one method to mitigate this without changing the downstream Transformer (e.g., data augmentation or invariant processing), and briefly justify it.

**What to submit?** 1-2 sentences.

★ **Solution** ★

## 1.3 Injecting Edge Features into Attention [4 points]

Suppose $G$ has edge features $\mathbf{e}_{ij} \in \mathbb{R}^m$ for $(i,j) \in E$. A practical way to incorporate them is to *bias* the attention logits:

$$\tilde{\alpha}_{ij} \propto \exp\Big(\mathbf{q}_i^\top \mathbf{k}_j + c_{ij}\Big), \qquad c_{ij} = \begin{cases} \mathbf{w}_1^\top \mathbf{e}_{ij}, & (i,j) \in E \\ \sum_{r=1}^{R} \sigma\big(\mathbf{w}_r^\top \mathbf{e}^{(r)}\big), & \text{if } j \text{ is at path-length } r \le R \end{cases}$$

where $\{\mathbf{e}^{(r)}\}$ are features along a shortest path from $i$ to $j$ of length $r$ (if within radius $R$).

(a) In 1–2 sentences, explain how $c_{ij}$ changes the receptive field and what happens if $R = 1$ vs. larger $R$.

(b) Give one advantage and one drawback of this biasing scheme.

**What to submit?** 3–4 sentences total.

★ **Solution** ★

## 1.4 Complexity and Sparsity Trade-offs [4 points]

Let $n$ be the number of nodes, $d$ the model hidden width per token, and let the graph have $|E|$ edges. Unless otherwise noted, count only the attention-specific work (score computation and attention–value product); the linear projections for $Q, K, V$ cost $O(nd^2)$ in both cases and do not change the asymptotic comparison.

(a) State the per-layer time complexity for:

- Dense self-attention run over all $n$ node tokens.
- Sparse, masked attention like we showed in Problem 1.1 where each node attends only to its neighbors (i.e., edges in $E$).

(b) In one sentence each, give:

- A setting where dense attention is preferable.
- A setting where masked sparse attention is preferable.

**What to submit?** The two complexities and two sentences.

★ **Solution** ★

## 2 LightGCN [13 points]

*Coverage: This problem is covered in Lecture 11.*

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 $k \in \{0, 1, ..., K-1\}$:

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

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

$$\mathbf{E} = \sum_{i=0}^{K} \alpha_i \mathbf{E}^{(i)}, \tag{2}$$

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

### 2.1 Advantages of Average Embeddings [4 points]

Why does LightGCN aerage 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 $\beta$ 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 (2), we can see that by setting $\alpha_k$ 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 $\alpha$ properly, LightGCN allows using a large K for long-range modeling with controllable oversmoothing.

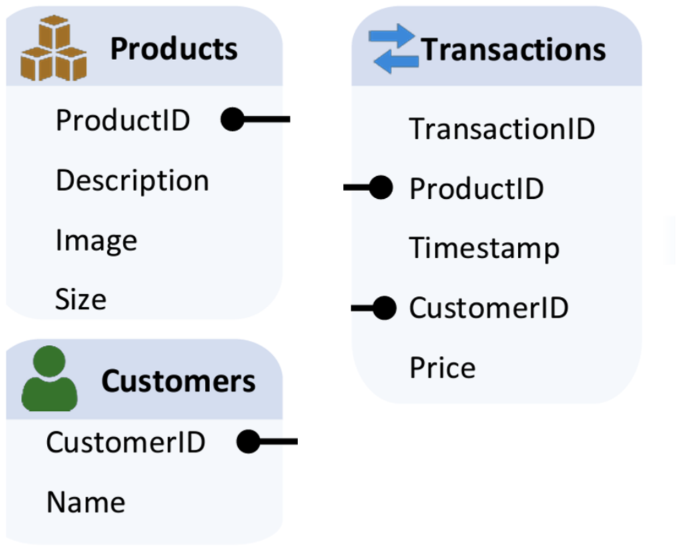
Express the layer-$K$ embeddings $\mathbf{E}^{(K)}$ of APPNP as a function of the initial embeddings $\mathbf{E}^{(0)}$ and the normalized adjacency matrix $\tilde{\mathbf{A}}$. Show all work.

**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]

*Coverage: This problem is covered in Lecture 12.*



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 in their respective tables, while `ProductID` and `CustomerID` are also foreign keys for the `Transactions` table.

## 3.1 Schema Graph [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 what the schema graph of this database would look like or draw it (hint: it's very simple).

★ **Solution** ★

## 3.2 Relational Entity Graph [4 points]

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 below, produce a relational entity graph describing this database.

In your graph, nodes should be labeled by their primary key values (for example, `102` rather than `Bob`).

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 | 1300 |
| 1004 | 4 | 2024-11-01 | 101 | 100 |
| 1005 | 1 | 2024-11-02 | 101 | 600 |
| 1006 | 2 | 2024-11-12 | 103 | 1300 |

★ **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 on `2024-10-20`, `2024-11-01`, and `2024-11-12`.

You should provide a computation graph for each prediction date. Note that since we want to make predictions on the *customer* level, you'll need to determine how edges in your computation graphs should be directed (i.e., the direction of message passing).

★ **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? We will accept any reasonable message passing and update rules for a heterogenous GNN that could model the data.

★ **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!**