DSC250: Advanced Data Mining

Recommeder System

Zhiting Hu

Lecture 17, November 28, 2023

UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

Logistics

Outline

- Recommender System

- 5 paper presentations
  - Gabriel Pila, Vivek S
  - Sai Kaushik Soma, Harsha Vardhan gangala
  - Barry Xiong, Fei Teng
  - Ishita Khatri, Yashi Shukla
  - Swetha Arunraj, Mohammed Alblooshi

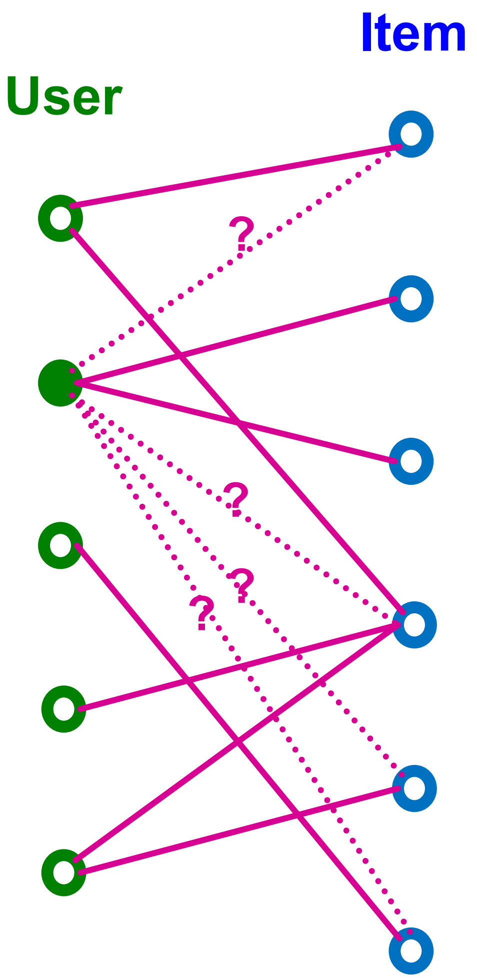
Recommender System (RecSys)

Slides adapted from:

- Y. Sun, CS 247: Advanced Data Mining
- Jure Leskovec, Stanford CS224W: Machine Learning with Graphs

Recommendation as Link Prediction

- **Given**
  - Past user-item interactions
- **Task**
  - Predict new items each user will interact in the future.
  - Can be cast as **link prediction** problem.
    - Predict new user-item interaction edges given the past edges.
  - For $u \in \boldsymbol{U}, v \in \boldsymbol{V}$, we need to get a real-valued **score** $f(u, v)$.

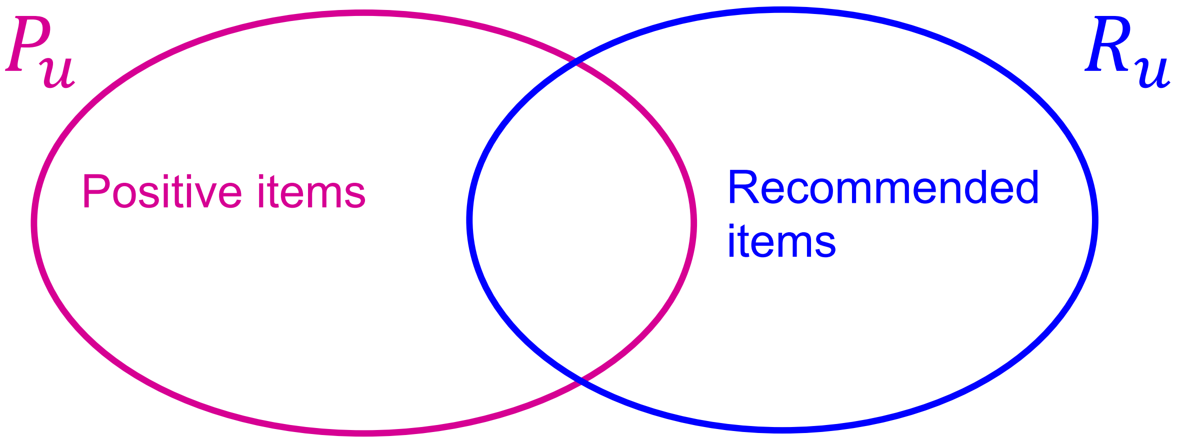


Top-K Recommendation

- For each user, we recommend K items.
  - **For recommendation to be effective, K needs to be much smaller than the total number of items (up to billions)**
  - K is typically in the order of 10—100.
- The goal is to include as many **positive items** as possible in the top-K recommended items.
  - **Positive items = Items that the user will interact with in the future.**
- **Evaluation metric**: Recall@K (defined next)

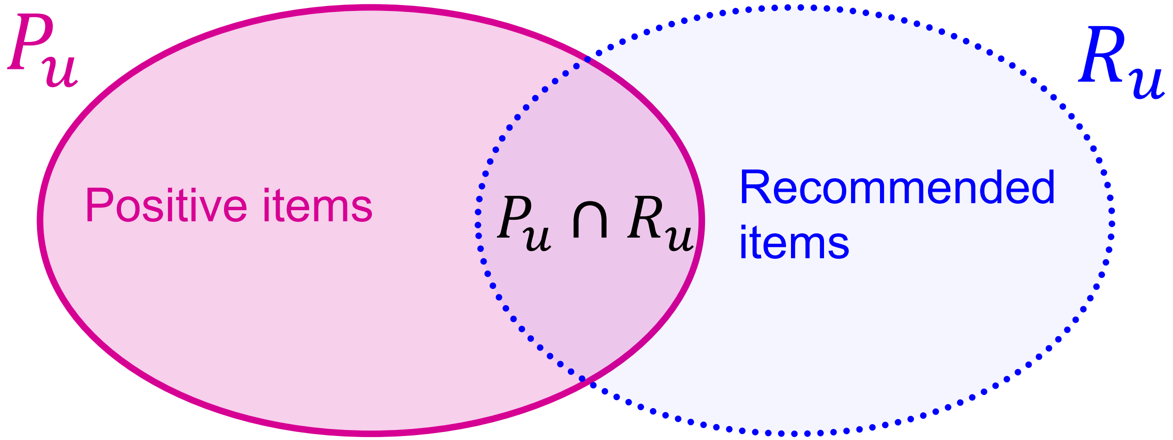
Evaluation Metric: Recall@K

- **For each user $\boldsymbol{u}$,**
  - Let P_u be a set of positive items the user will interact in the future.
  - Let R_u be a set of items recommended by the model.
    - In top-K recommendation, $|R_u| = K$.
    - Items that the user has already interacted are excluded.



Evaluation Metric: Recall@K

- **Recall@K for user u is $|P_u \cap R_u| / |P_u|$.**
  - Higher value indicates more positive items are recommended in top-K for user u.



- The final Recall@K is computed by averaging the recall values across all users.

Methods

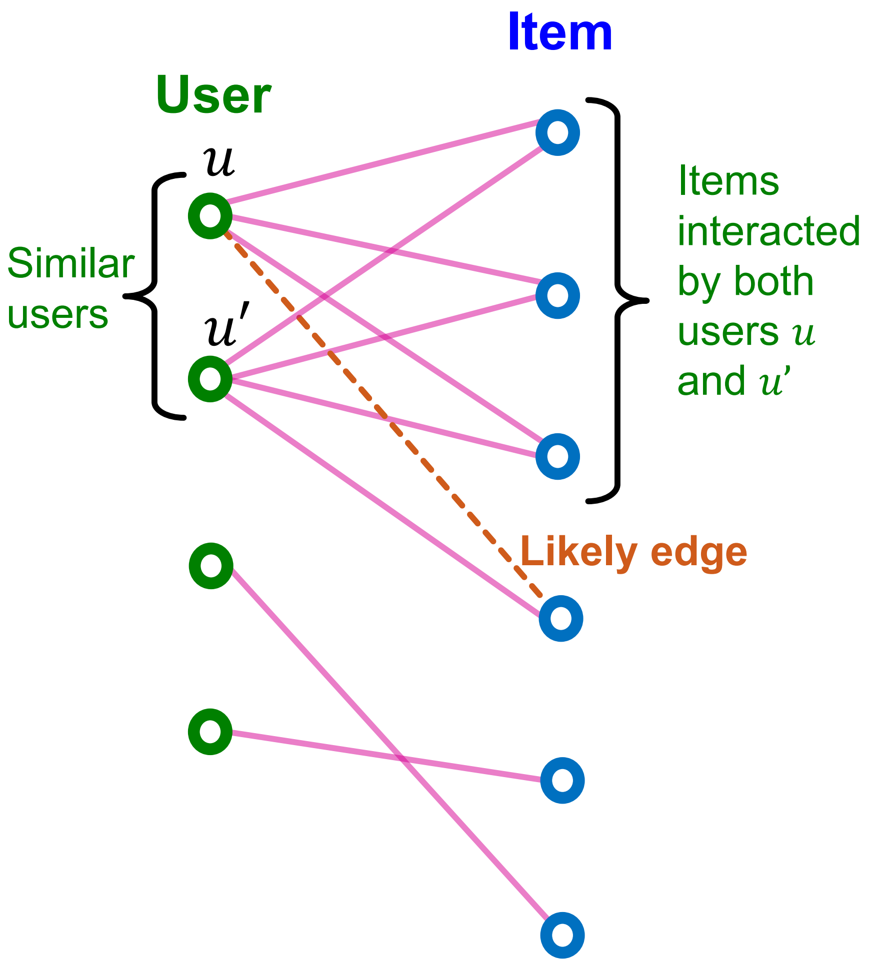
- Collaborative filtering
- Content-based recommendation
- Hybrid methods

Methods

- Collaborative filtering
- Content-based recommendation
- Hybrid methods

Collaborative Filtering (CF)

- **Underlying idea: Collaborative filtering**
  - Recommend items for a user by **collecting preferences of many other similar users.**
  - **Similar users tend to prefer similar items.**
- **Key question: How to capture similarity between users/items?**



Collaborative Filtering (CF): Methods

- Memory-based Collaborative Filtering
  - User-based CF
    - Compute similarity between users and active users, and use similar users' ratings as prediction
  - Item-based CF
    - Compute similarity between items, and predict similar rating to similar items that the active user has rated before
- Model-based Collaborative Filtering

Collaborative Filtering (CF): Methods

- Memory-based Collaborative Filtering
  - User-based CF
    - Compute similarity between users and active users, and use similar users' ratings as prediction
  - Item-based CF
    - Compute similarity between items, and predict similar rating to similar items that the active user has rated before
- Model-based Collaborative Filtering

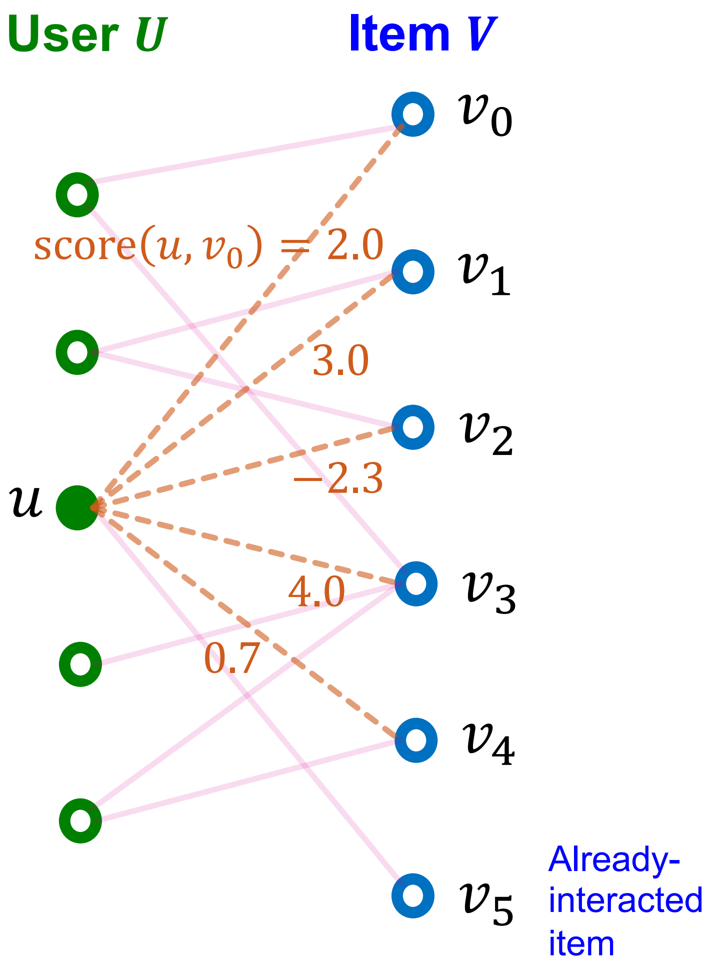
- The rating matrix is directly used to find neighbors / make predictions
- Does not scale for most real-world scenarios

Collaborative Filtering (CF): Methods

- Memory-based Collaborative Filtering
- Model-based Collaborative Filtering
  - based on an offline pre-processing or "model-learning" phase
  - at run-time, only the learned model is used to make predictions
  - models are updated / re-trained periodically
  - large variety of techniques used
  - model-building and updating can be computationally expensive

Embedding-Based Models

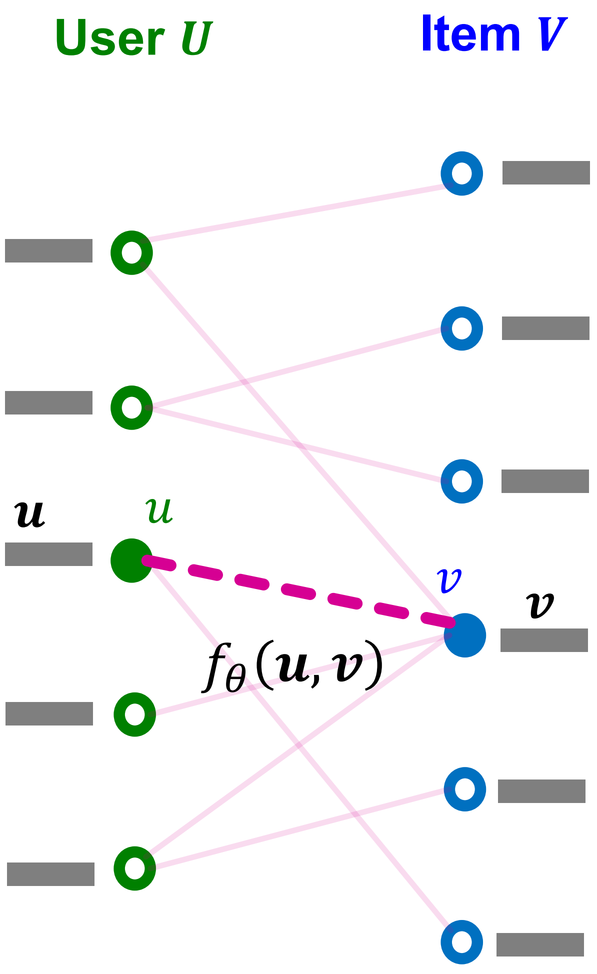
- To get the top-K items, we need a score function for user-item interaction:
  - For $u \in \boldsymbol{U}, v \in \boldsymbol{V}$, we need to get a real-valued scalar **score**(u, v).
  - ***K* items with the largest scores for a given user u** (excluding already-interacted items) are then recommended.



For $K = 2$, recommended items for user u would be $\{v_1, v_3\}$.

Embedding-Based Models

- We consider **embedding-based models** for scoring user-item interactions.
  - For each user $u \in \boldsymbol{U}$, let $\boldsymbol{u} \in \mathbb{R}^D$ be its D-dimensional embedding.
  - For each item $v \in \boldsymbol{V}$, let $\boldsymbol{v} \in \mathbb{R}^D$ be its D-dimensional embedding.
  - Let $f_\theta(\cdot,\cdot): \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$ be a parametrized function.
  - Then, $\boxed{\text{score}(u, v) \equiv f_\theta(\boldsymbol{u}, \boldsymbol{v})}$



Embedding-Based Models: Training Objective

- Embedding-based models have three kinds of parameters:
  - An encoder to generate user embeddings $\{\boldsymbol{u}\}_{u \in U}$
  - An encoder to generate item embeddings $\{\boldsymbol{v}\}_{v \in V}$
  - Score function $f_\theta(\cdot,\cdot)$
- **Training objective**: Optimize the model parameters to achieve high recall@K on *seen* (i.e., *training*) user-item interactions
  - We hope this objective would lead to high recall@K on *unseen* (i.e., *test*) interactions.

Embedding-Based Models: Surrogate Loss Functions

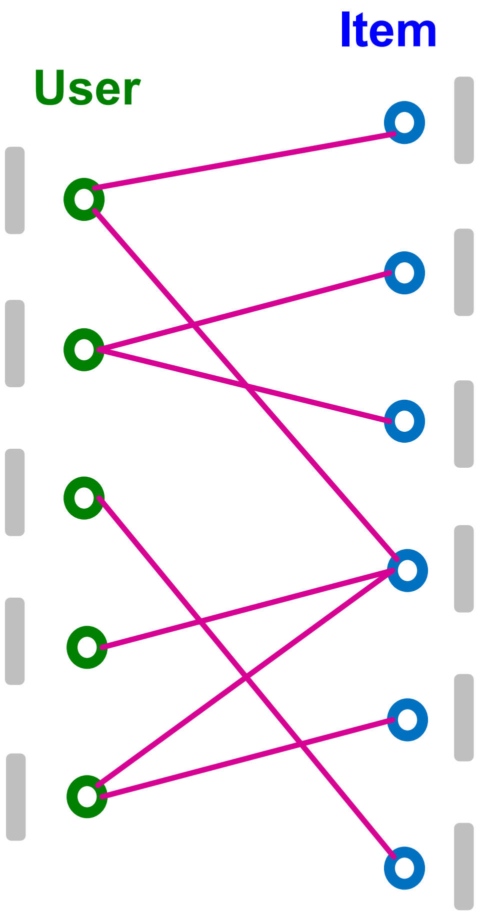
- The original training objective (recall@K) is ***not* differentiable**.
  - *Cannot* apply efficient gradient-based optimization.
- Two **surrogate loss functions** are widely-used to enable efficient gradient-based optimization.
  - Binary loss
  - Bayesian Personalized Ranking (BPR) loss
- Surrogate losses are **differentiable** and should **align well with the original training objective**.

Why Embedding Models Work?

- Embedding-based models can capture similarity of users/items!
  - **Low-dimensional embeddings *cannot* simply memorize all user-item interaction data.**
  - Embeddings are forced to **capture similarity between users/items to fit the data.**
  - This allows the models to make effective prediction on *unseen* user-item interactions.

Conventional Embedding-based CF

- Conventional collaborative filtering model is based on **shallow encoders**:
  - Use shallow encoders for users and items:
    - For every $u \in \boldsymbol{U}$ and $v \in \boldsymbol{V}$, we prepare shallow learnable embeddings $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^D$.
  - Score function for user u and item v is $f_\theta(\boldsymbol{u}, \boldsymbol{v}) \equiv \boldsymbol{z_u}^T \boldsymbol{z_v}$.



Learnable shallow user/item embeddings

Limitations of Shallow Encoders

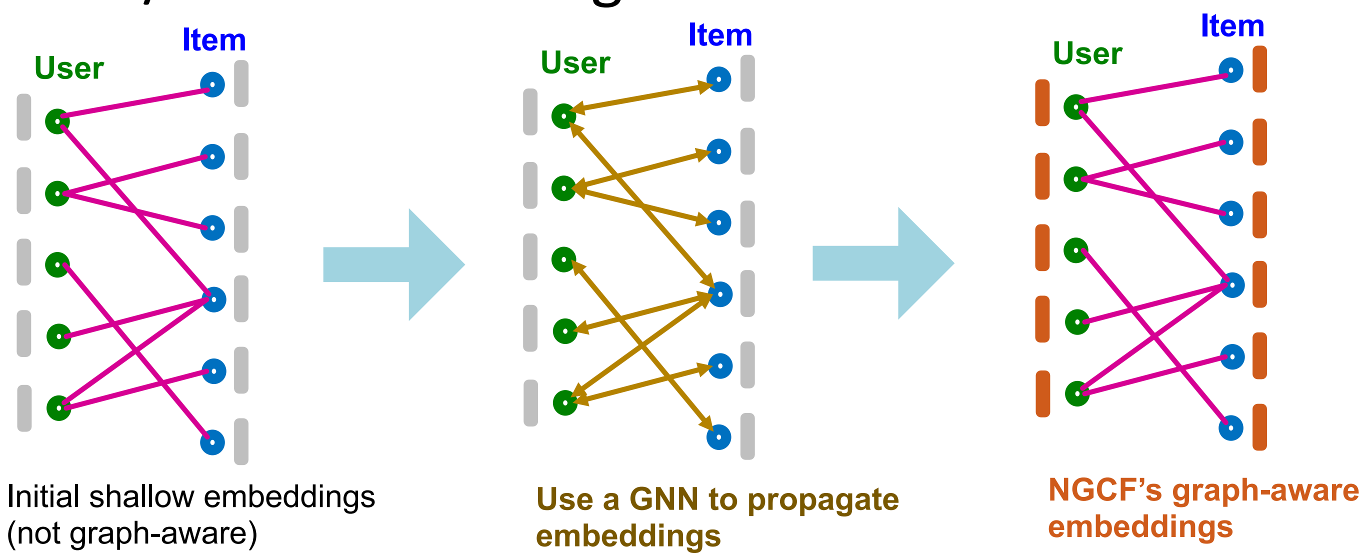
- The model itself does ***not explicitly*** capture graph structure
  - The graph structure is ***only implicitly*** captured in the training objective.
- Only the **first-order graph structure** (i.e., edges) is captured in the training objective.
  - **High-order graph structure** (e.g., K-hop paths between two nodes) is ***not* explicitly captured.**

We want a model that …

- We want a model that…
  - **explicitly captures graph structure** (beyond implicitly through the training objective)
  - captures **high-order graph structure** (beyond the first-order edge connectivity structure)
- **GNNs are a natural approach to achieve both!**
  - **Neural Graph Collaborative Filtering** (NGCF) [Wang et al. 2019]
  - **LightGCN** [He et al. 2020]
    - A simplified and improved version of NGCF

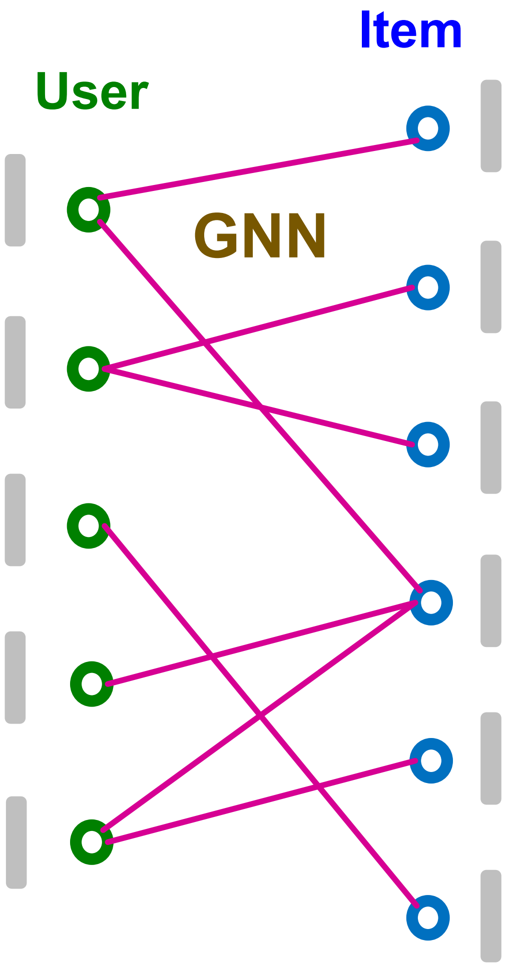
NGCF: Overview

- **Neural Graph Collaborative Filtering (NGCF)** ***explicitly*** incorporates high-order graph structure when generating user/item embeddings.
- **Key idea**: Use a GNN to generate graph-aware user/item embeddings.



NGCF

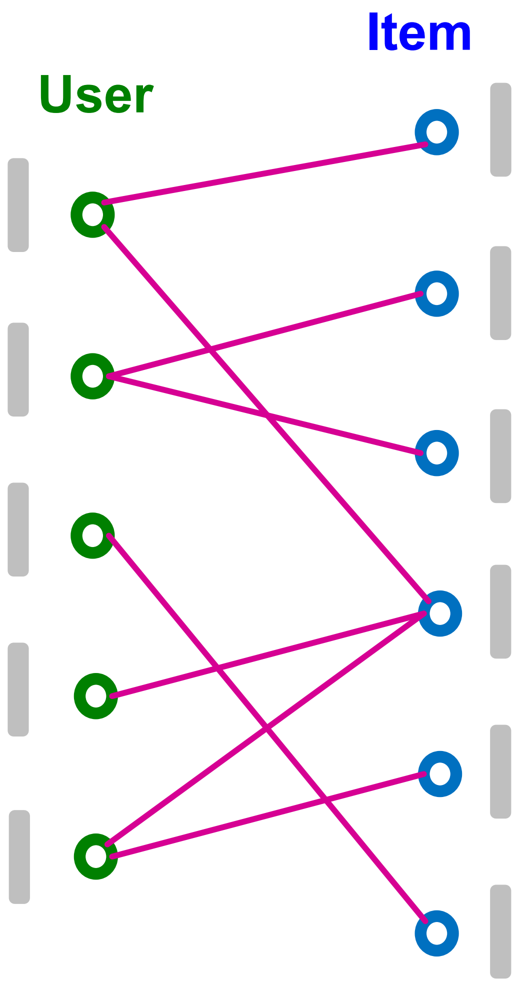
- **Given**: User-item bipartite graph.
- **NGCF framework:**
  - Prepare shallow learnable embedding for each node.
  - Use multi-layer GNNs to propagate embeddings along the bipartite graph.
    - High-order graph structure is captured.
  - Final embeddings are *explicitly* graph-aware!
- **Two kinds of learnable params are jointly learned:**
  - Shallow user/item embeddings
  - GNN's parameters



Shallow user/item embeddings (learnable)

NGCF: Initial Node Embeddings

- Set the shallow learnable embeddings as the initial node features:
  - For every user $u \in \boldsymbol{U}$, set $\boldsymbol{h}_u^{(0)}$ as the user's shallow embedding.
  - For every item $v \in \boldsymbol{V}$, set $\boldsymbol{h}_v^{(0)}$ as the item's shallow embedding.



Learnable shallow user/item embeddings

NGCF: Neighbor Aggregation

- Iteratively update node embeddings using neighboring embeddings.

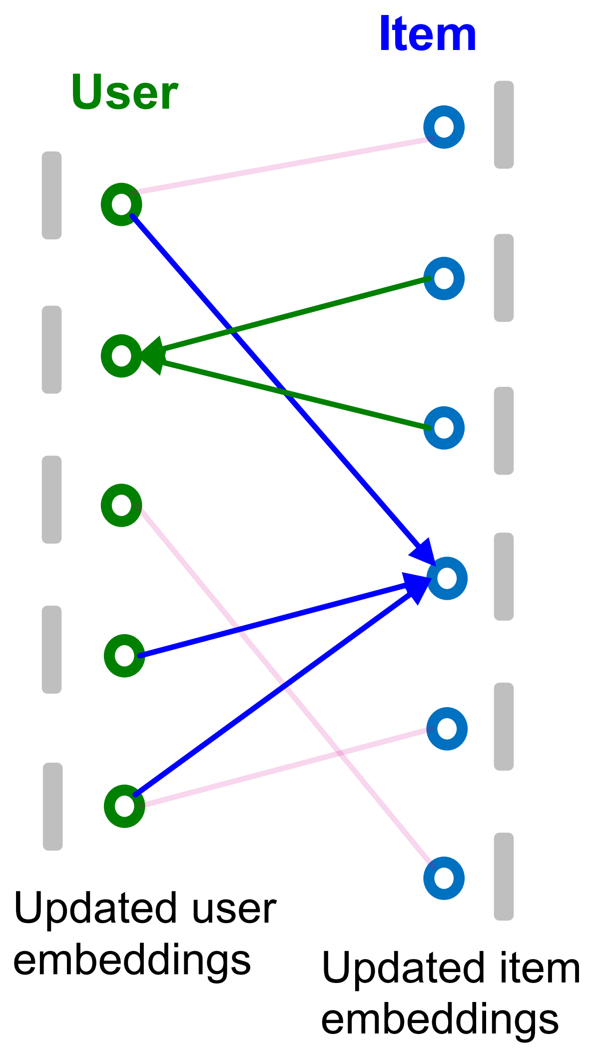
$$\boldsymbol{h}_v^{(k+1)} = \mathrm{COMBINE}\left(\boldsymbol{h}_v^{(k)}, \mathrm{AGGR}\left(\left\{\boldsymbol{h}_u^{(k)}\right\}_{u \in N(v)}\right)\right)$$

$$\boldsymbol{h}_u^{(k+1)} = \mathrm{COMBINE}\left(\boldsymbol{h}_u^{(k)}, \mathrm{AGGR}\left(\left\{\boldsymbol{h}_v^{(k)}\right\}_{v \in N(u)}\right)\right)$$

High-order graph structure is captured through iterative neighbor aggregation.

Different architecture choices are possible for AGGR and COMBINE.

- $\mathrm{AGGR}(\cdot)$ can be $\mathrm{MEAN}(\cdot)$
- $\mathrm{COMBINE}(\boldsymbol{x}, \boldsymbol{y})$ can be $\mathrm{ReLU}(\mathrm{Linear}(\mathrm{Concat}(\boldsymbol{x}, \boldsymbol{y})))$



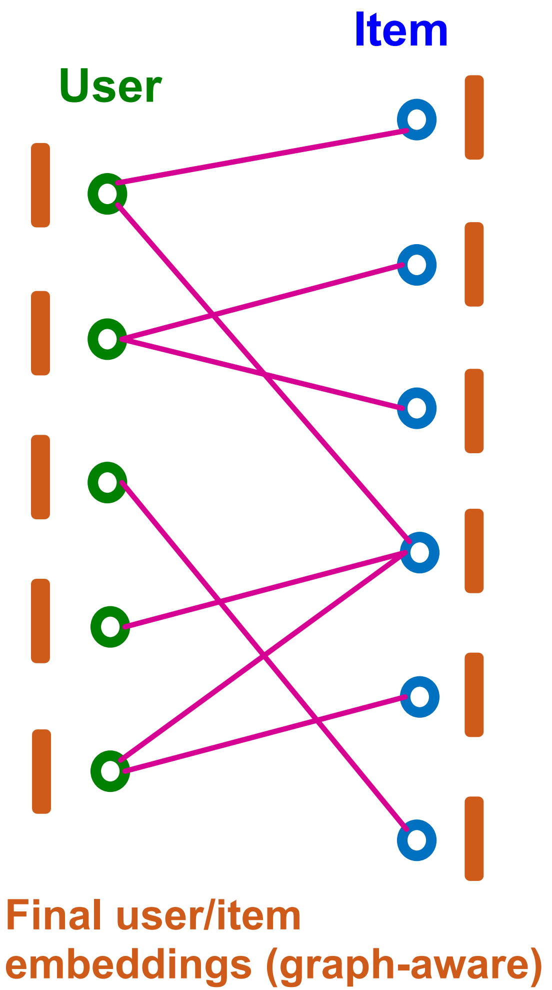
NGCF: Final Embeddings and Score Function

- After K rounds of neighbor aggregation, we get the **final user/item embeddings** $\boldsymbol{h}_u^{(K)}$ and $\boldsymbol{h}_v^{(K)}$.
- For all $u \in \boldsymbol{U}, v \in \boldsymbol{V}$, we set

$$\boldsymbol{u} \leftarrow \boldsymbol{h}_u^{(K)}, \boldsymbol{v} \leftarrow \boldsymbol{h}_v^{(K)}.$$

- Score function is the inner product

$$\text{score}(u, v) = \boldsymbol{u}^T \boldsymbol{v}$$



NGCF: Summary

- Conventional collaborative filtering uses shallow user/item embeddings.
  - The embeddings do ***not explicitly* model graph structure**.
  - The training objective **does not model high-order graph structure.**
- **NGCF uses a GNN to propagate the shallow embeddings.**
  - The embeddings are **explicitly aware of high-order graph structure.**

Issues of Collaborative Filtering

- **Cold Start**: There needs to be enough other users already in the system to find a match.
- **Sparsity**: If there are many items to be recommended, even if there are many users, the user/ratings matrix is sparse, and it is hard to find users that have rated the same items.
- **First Rater**: Cannot recommend an item that has not been previously rated.
  - New items
  - Esoteric items
- **Popularity Bias**: Cannot recommend items to someone with unique tastes.
  - Tends to recommend popular items.

Methods

- Collaborative filtering
- Content-based recommendation
- Hybrid methods

Content-based Recommendation

- Collaborative filtering does NOT require any information about content,
  - However, it might be reasonable to exploit such information
  - E.g. recommend fantasy novels to people who liked fantasy novels in the past
- What do we need:
  - Information about the available items such as the genre ("content")
  - *user profile* describing what the user likes (the preferences)
- The task:
  - Learn user preferences
  - Locate/recommend items that are "similar" to the user preferences

Content-based Recommendation

User profile

Title	Genre	Author	Type	Price	Keywords
The Night of the Gun	Memoir	David Carr	Paperback	29.90	Press and journalism, drug addiction, personal memoirs, New York
The Lace Reader	Fiction, Mystery	Brunonia Barry	Hardcover	49.90	American contemporary fiction, detective, historical
Into the Fire	Romance, Suspense	Suzanne Brockmann	Hardcover	45.90	American fiction, Murder, Neo-nazism
...					

Item

Title	Genre	Author	Type	Price	Keywords
...	Fiction, Suspense	Brunonia Barry, Ken Follet, ..	Paperback	25.65	detective, murder, New York

- Simple approach
  - Compute the similarity of an unseen item with the user profile based on the keyword overlap (e.g. using the Dice coefficient)
  - $\text{sim}(b_i, b_j) = \frac{2 * |keywords(b_i) \cap keywords(b_j)|}{|keywords(b_i)| + |keywords(b_j)|}$
- Other advanced similarity measure

Methods

- Collaborative filtering

- Content-based recommendation

- Hybrid methods
  - Combining both user-item interaction and other external sources of information
  - E.g., Factorization Machines

Questions?