

# GNN Applications: Recommender Systems

**Jiaxuan You**

**Assistant Professor at UIUC CDS**



**CS598: Deep Learning with Graphs, 2024 Fall**

**<https://ulab-uiuc.github.io/CS598/>**

GNN Applications: Recommender Systems

# Recommender Systems: Task and Evaluation

# Preliminary of Recommendation

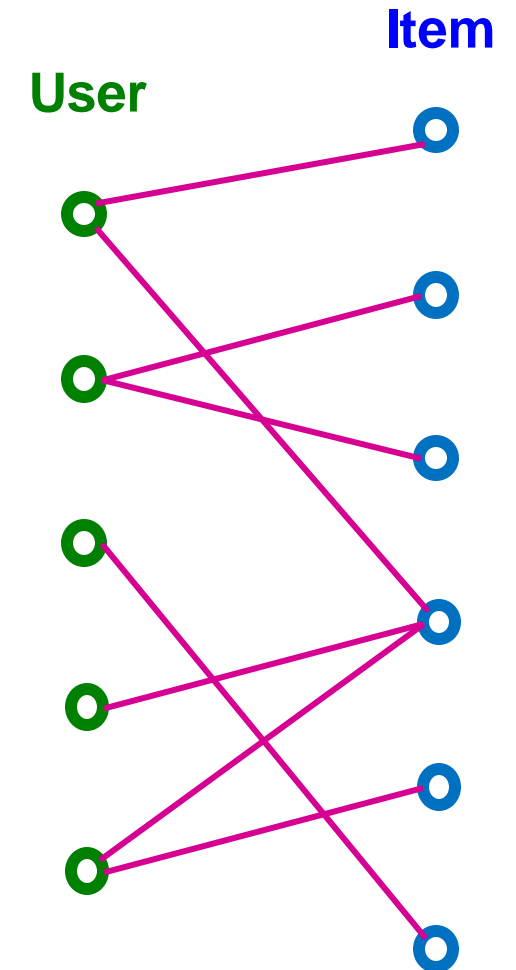
- **Information Explosion in the era of Internet**
  - 10K+ movies in Netflix
  - 12M products in Amazon
  - 70M+ music tracks in Spotify
  - 10B+ videos on YouTube
  - 200B+ pins (images) in Pinterest
- **Personalized recommendation (i.e., suggesting a small number of interesting items for each user)** is critical for users to effectively explore the content of their interest.

# Notation

- **Notation:**
  - $U$ : A set of all users
  - $V$ : A set of all items
  - $E$ : A set of observed user-item interactions
    - $E = \{(u, v) \mid u \in U, v \in V, u \text{ interacted with } v\}$

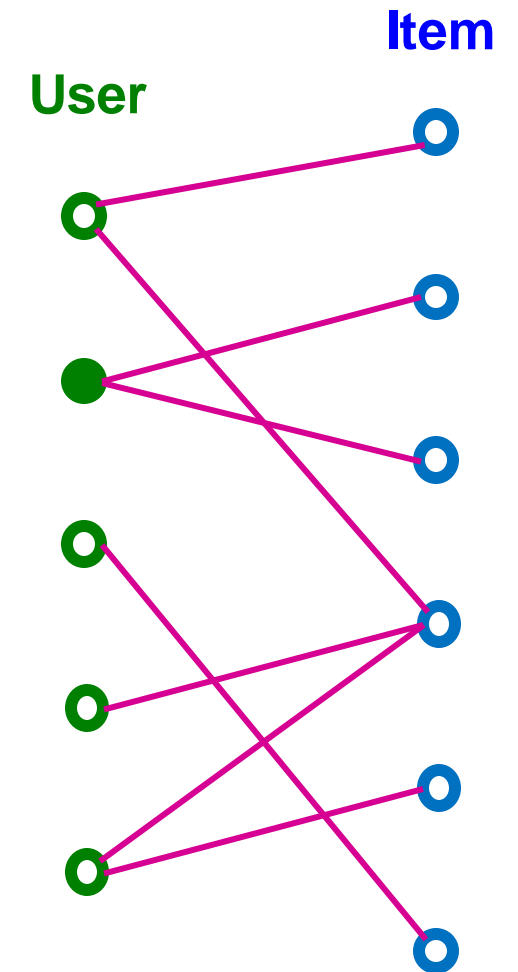
# Recommender System as a Graph

- Recommender system can be naturally modeled as a **bipartite graph**
  - A graph with two node types: **users** and **items**.
  - **Edges** connect users and items
    - Indicates user-item interaction (e.g., click, purchase, review etc.)
    - Often associated with timestamp (timing of the interaction).



# Recommendation Task

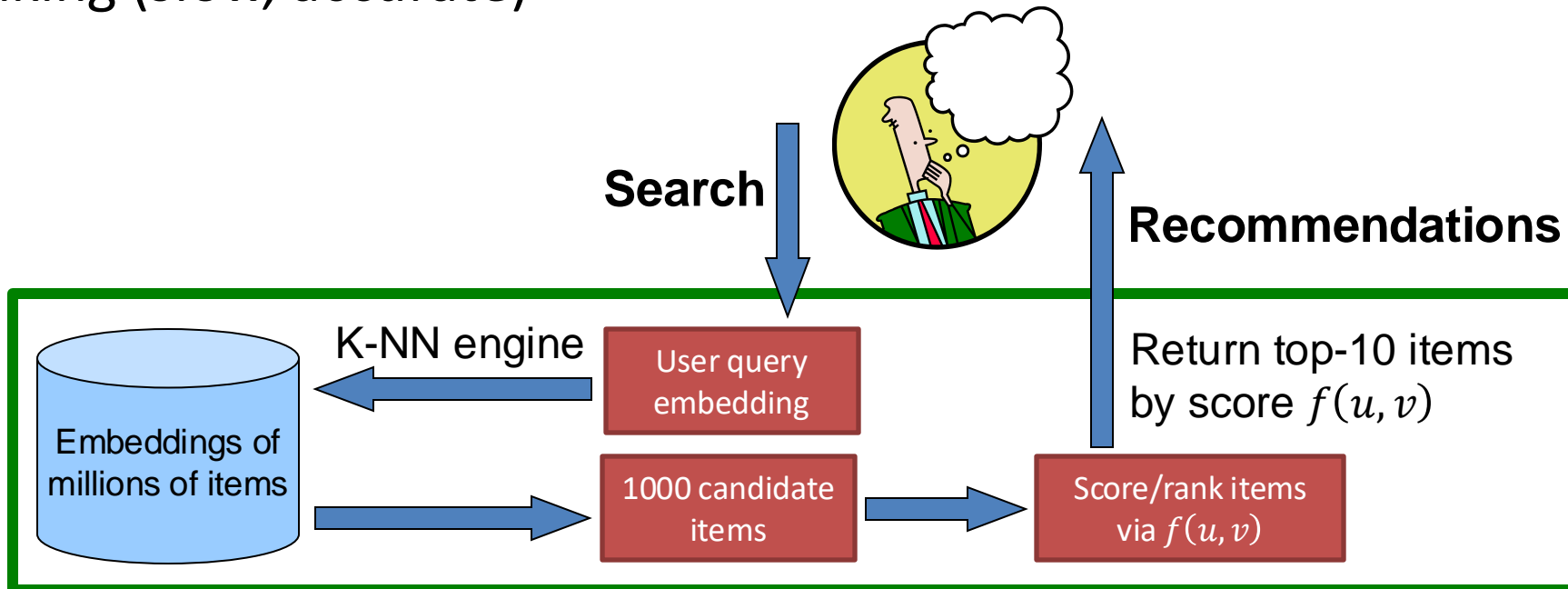
- **Given**
  - Past user-item interactions
- **Task**
  - Predict new items each user will interact in the future.
  - Can be cast as **link prediction** problem.
    - Predict new user-item interaction edges given the past edges.
  - For  $u \in U, v \in V$ , we need to get a real-valued **score**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



# Modern Recommender System

- **Problem:** Cannot evaluate  $f(u, v)$  for every user  $u$  – item  $v$  pair.
- **Solution:** 2-stage process:
  - Candidate generation (cheap, fast)
  - Ranking (slow, accurate)

Example  $f(u, v)$ :  
 $f(u, v) = z_u \cdot z_v$



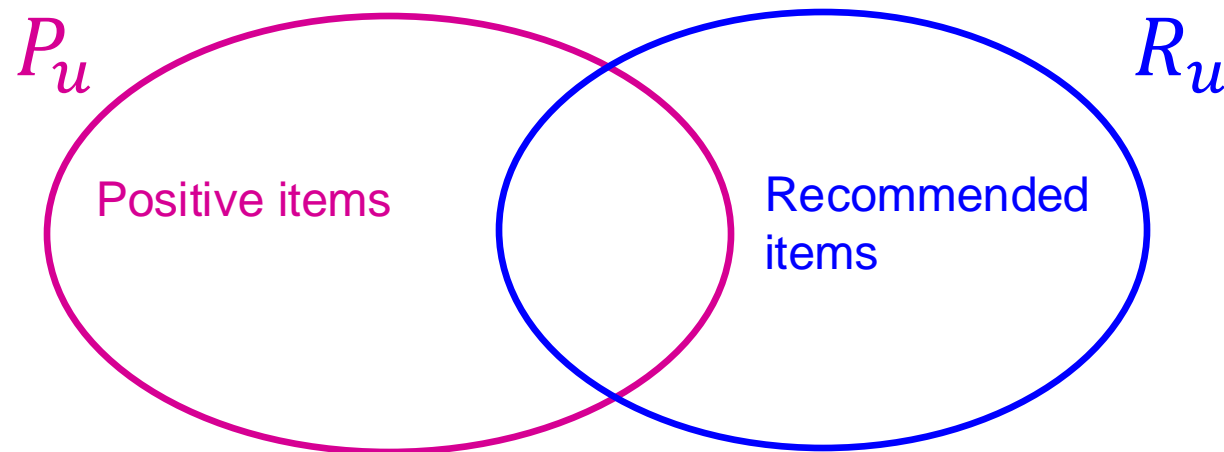
# Top- $K$ Recommendation

- 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)



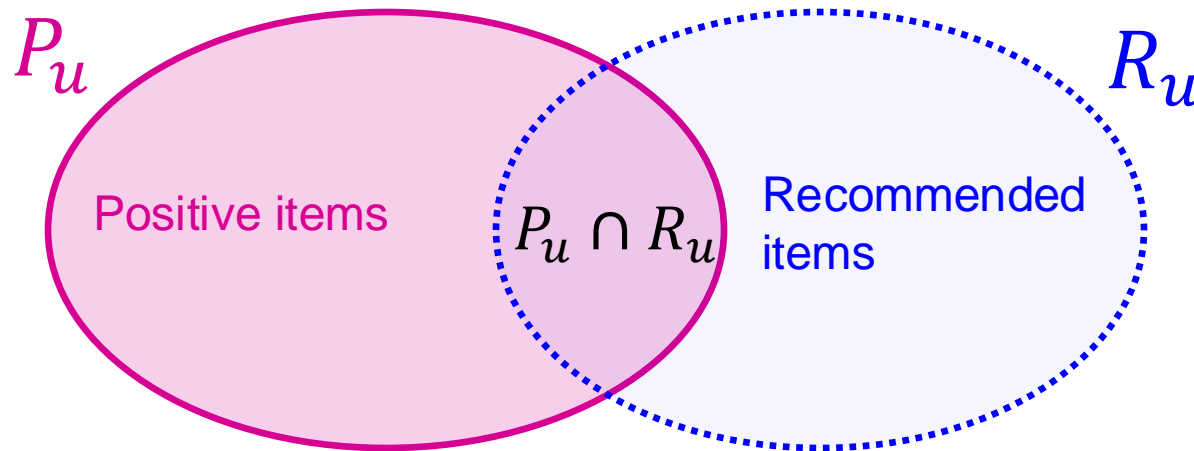
# Evaluation Metric: Recall@K (1)

- For each user  $u$ ,
  - 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 (2)

- **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  $u$ .



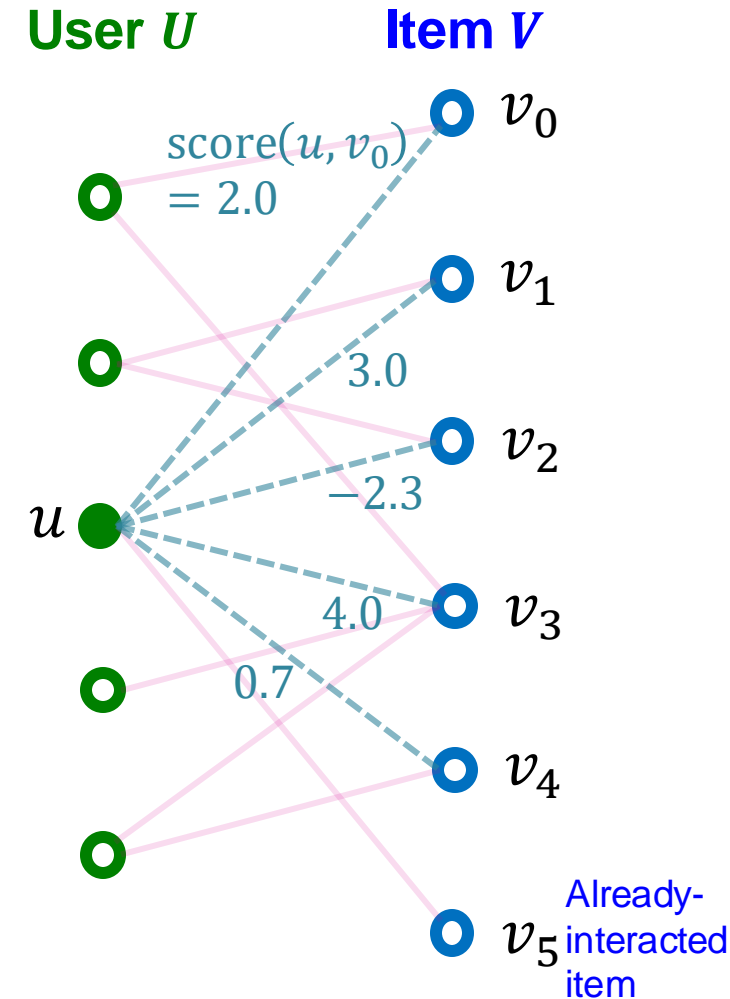
- The final Recall@K is computed by averaging the recall values across all users.

GNN Applications: Recommender Systems

# Recommender Systems: Embedding-Based Models

# Score Function

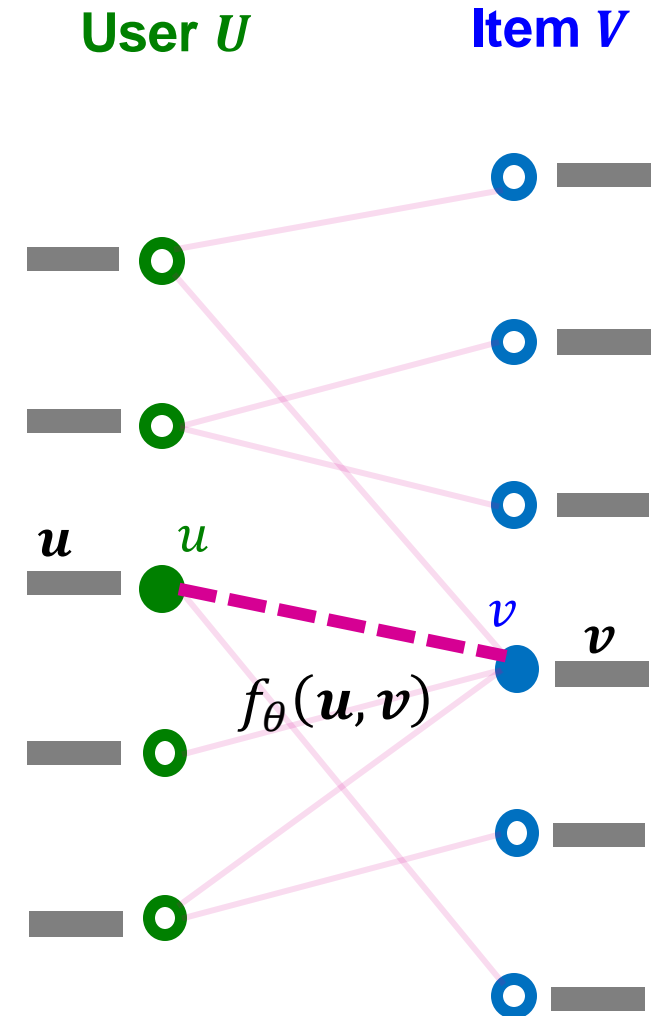
- To get the top- $K$  items, we need a score function for user-item interaction:
  - For  $u \in U, v \in V$ , we need to get a real-valued scalar **score**( $u, v$ ).
  - **$K$  items with the largest scores for a given user  $u$**  (excluding **already-interacted items**) are then recommended.



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  $u \in U$ , let  $\mathbf{u} \in \mathbb{R}^D$  be its  $D$ -dimensional embedding.
  - For each item  $v \in V$ , let  $\mathbf{v} \in \mathbb{R}^D$  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,  **$\text{score}(u, v) \equiv f_\theta(\mathbf{u}, \mathbf{v})$**



# Training Objective

- Embedding-based models have three kinds of parameters:
  - An encoder to generate user embeddings  $\{\mathbf{u}\}_{u \in U}$
  - An encoder to generate item embeddings  $\{\mathbf{v}\}_{v \in V}$
  - 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.

# Surrogate Loss Functions

- The original training objective ( $\text{recall}@K$ ) 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**.

# Loss Function: Binary Loss

- **Binary loss**: Binary classification of **positive**/**negative** edges using  $\sigma(f_\theta(\mathbf{u}, \mathbf{v}))$ :

$$-\underbrace{\frac{1}{|E|} \sum_{(u,v) \in E} \log(\sigma(f_\theta(\mathbf{u}, \mathbf{v})))}_{\text{positive edges}} - \underbrace{\frac{1}{|E_{\text{neg}}|} \sum_{(u,v) \in E_{\text{neg}}} \log(1 - \sigma(f_\theta(\mathbf{u}, \mathbf{v})))}_{\text{negative edges}}$$

During training, these terms can be approximated using mini-batch of positive/negative edges

- Binary loss pushes the scores of **positive edges** higher than those of **negative edges**.
  - This aligns with the training recall metric since positive edges need to be recalled.



# Loss Function: BPR Loss

- **Training objective:** For each user  $u^*$ , we want the scores of rooted positive edges  $E(u^*)$  to be higher than those of rooted negative edges  $E_{\text{neg}}(u^*)$ .
  - Aligns with the personalized nature of the recall metric.
- **BPR Loss for user  $u^*$ :**

Encouraged to be positive for each user

=positive edge score is higher than negative edge score

$$\text{Loss}(u^*) = \frac{1}{|E(u^*)| \cdot |E_{\text{neg}}(u^*)|} \underbrace{\sum_{(u^*, v_{\text{pos}}) \in E(u^*)} \sum_{(u^*, v_{\text{neg}}) \in E_{\text{neg}}(u^*)} -\log \left( \sigma \left( \overbrace{f_{\theta}(u^*, v_{\text{pos}}) - f_{\theta}(u^*, v_{\text{neg}})} \right) \right)}_{\text{Can be approximated using a mini-batch}}$$

Can be approximated using a mini-batch

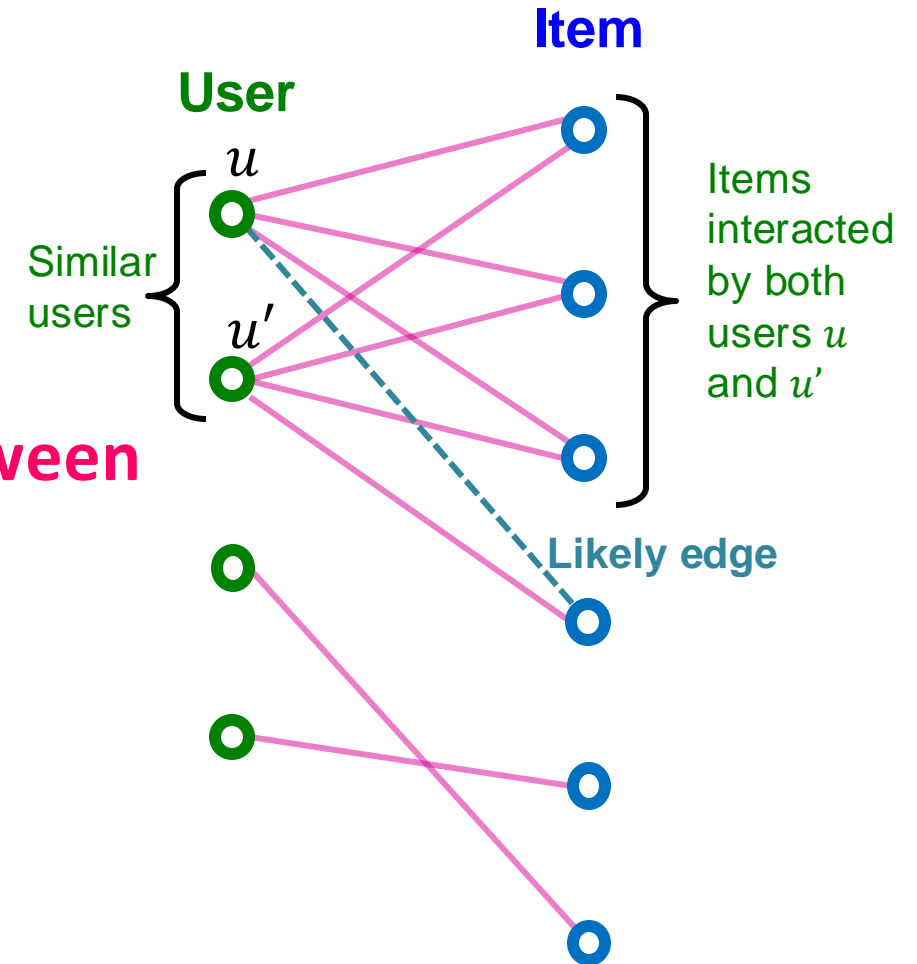
- Final BPR Loss:  $\frac{1}{|U|} \sum_{u^* \in U} \text{Loss}(u^*)$

# Summary So Far

- **We have introduced**
  - Recall@ $K$  as a metric for personalized recommendation
  - Embedding-based models
    - Three kinds of parameters to learn
      - **user encoder** to generate user embeddings
      - **item encoder** to generate item embeddings
      - **score function** to predict the user-item interaction likelihood.
    - Surrogate loss functions to achieve the high recall metric.
- Embedding-based models have achieved SoTA in recommender systems.
  - **Why do they work so well?**

# Why Embedding Models Work?

- **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?**



# 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.

# This Lecture: GNNs for Recsys

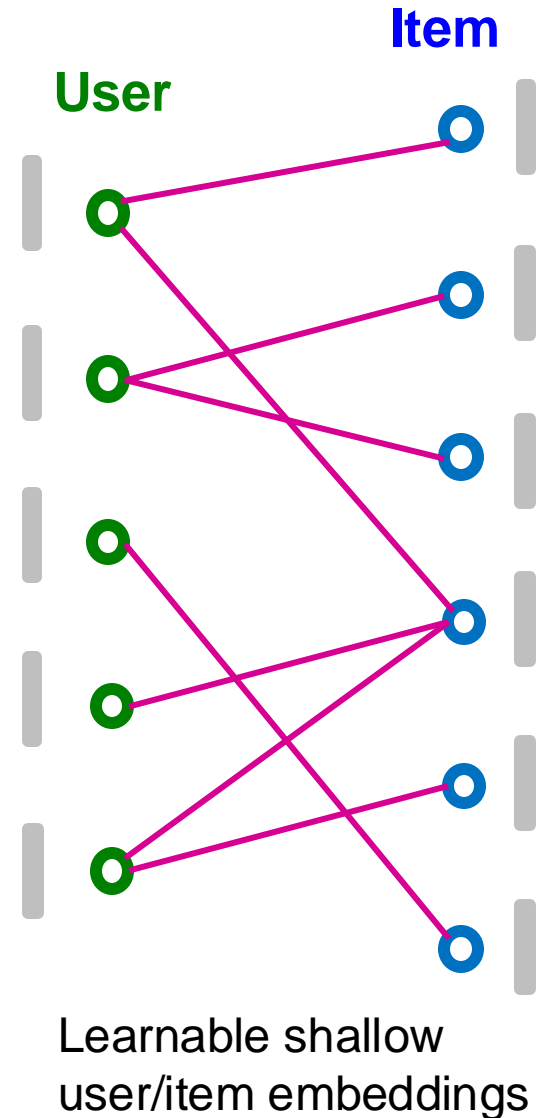
- In this lecture, we teach two representative GNN approaches for recommender systems.
- (1) Neural Graph Collab. Filtering (NGCF)
- (2) LightGCN [He et al. 2020]
  - Improve the conventional collaborative filtering models (i.e., shallow encoders) by explicitly modeling graph structure using GNNs.
  - Assumes no user/item features.
- PinSAGE [Ying et al. 2018]
  - Use GNNs to generate high-quality embeddings by simultaneously capturing rich node attributes (e.g., images) and the graph structure.

GNN Applications: Recommender Systems

# Neural Graph Collaborative Filtering

# Conventional Collaborative Filtering

- Conventional collaborative filtering model is based on **shallow encoders**:
  - No user/item features.
  - Use shallow encoders for users and items:
    - For every  $u \in U$  and  $v \in V$ , we prepare shallow learnable embeddings  $\mathbf{u}, \mathbf{v} \in \mathbb{R}^D$ .
  - Score function for user  $u$  and item  $v$  is  $f_{\theta}(\mathbf{u}, \mathbf{v}) \equiv \mathbf{z}_u^T \mathbf{z}_v$ .



# 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*.

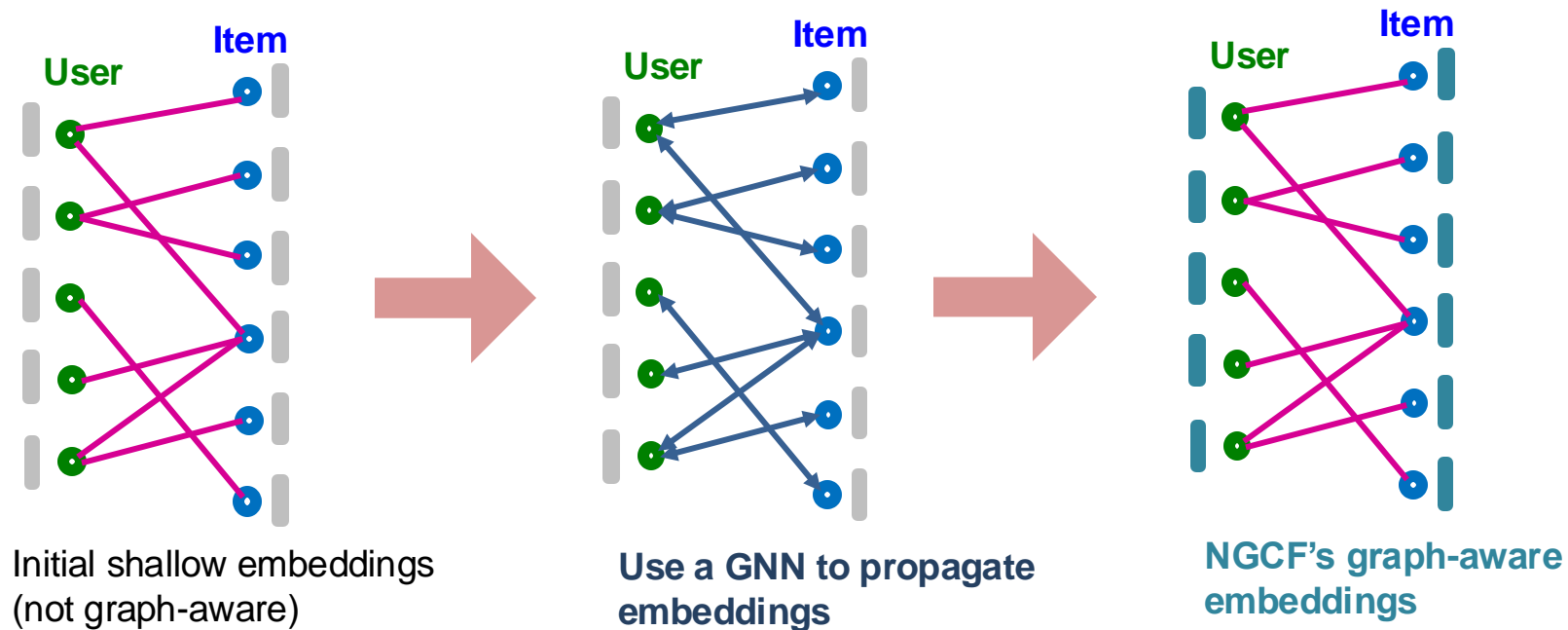


# Motivation

- 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

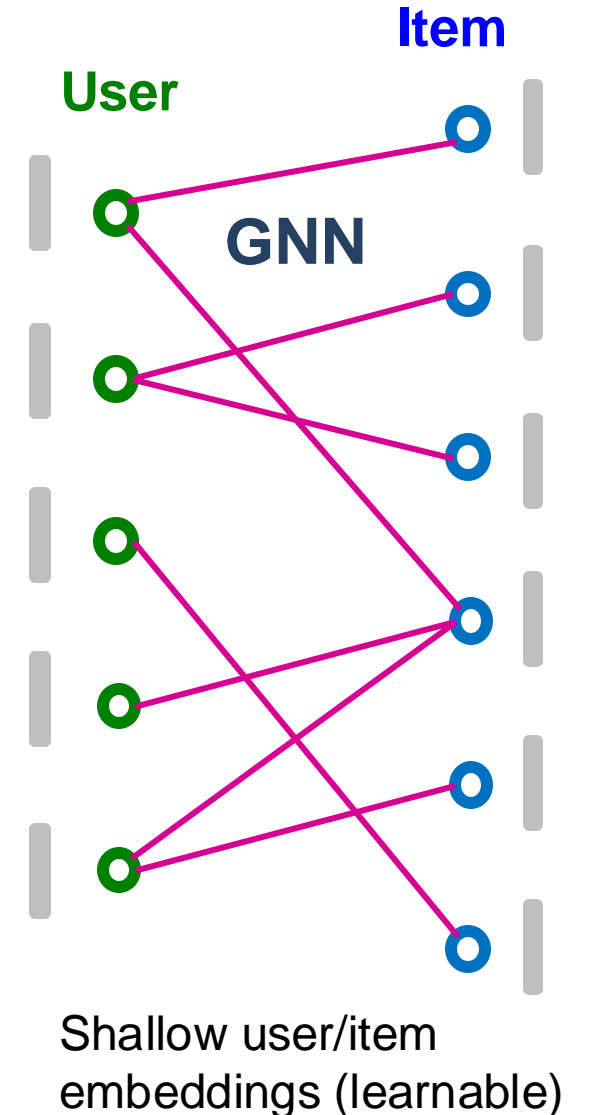
# NGCF: Overview

- **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.



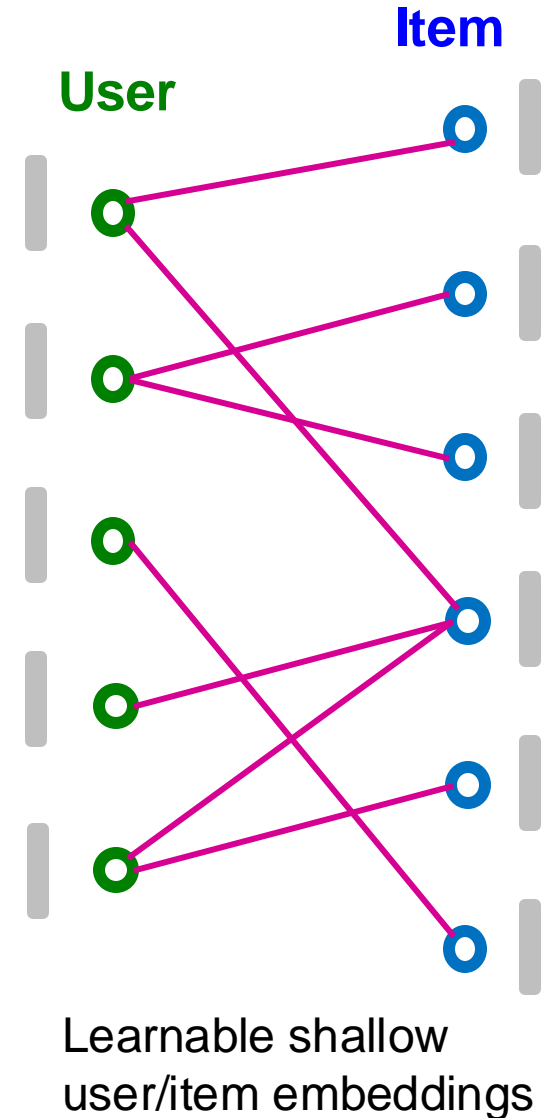
# NGCF Framework

- **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



# Initial Node Embeddings

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



# Neighbor Aggregation

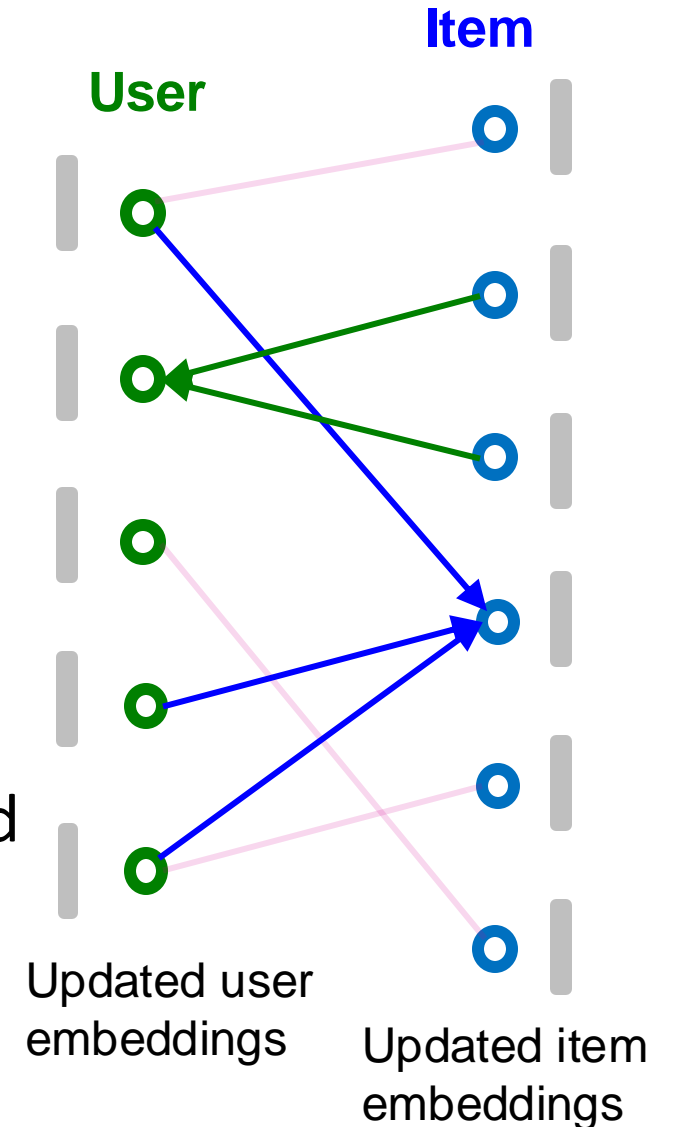
- Iteratively update node embeddings using neighboring embeddings.

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

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

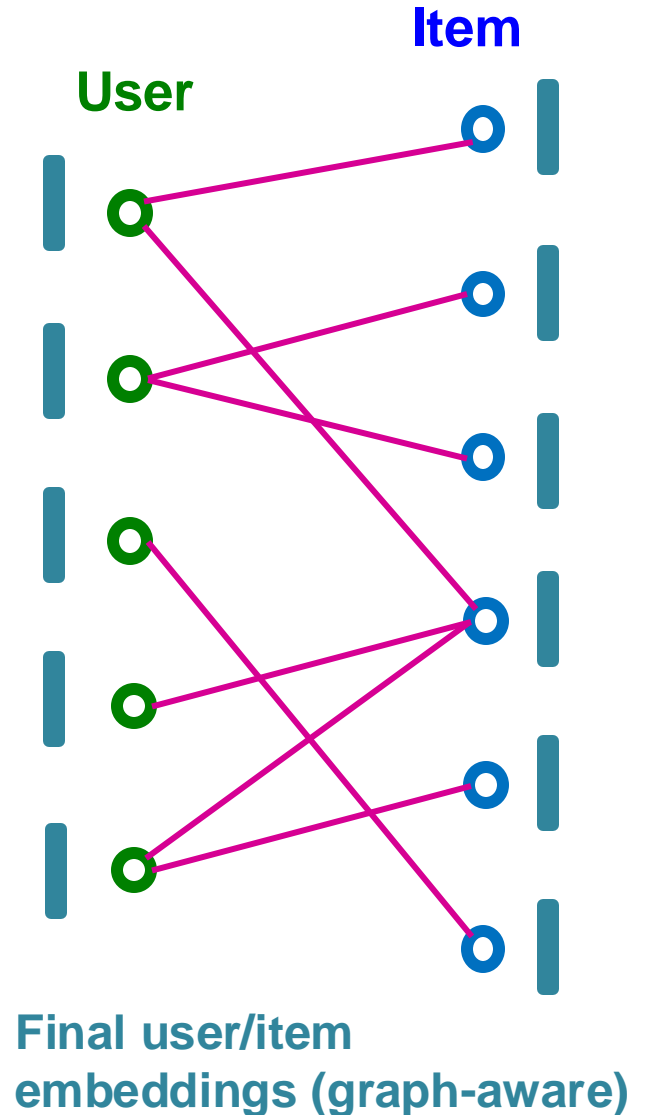
**High-order graph structure is captured through iterative neighbor aggregation.**

- Different architecture choices are possible for AGGR and COMBINE.
- AGGR( $\cdot$ ) can be MEAN( $\cdot$ )
- COMBINE( $\mathbf{x}, \mathbf{y}$ ) can be  $\text{ReLU}(\text{Linear}(\text{Concat}(\mathbf{x}, \mathbf{y})))$



# Final Embeddings and Score Function

- After  $K$  rounds of neighbor aggregation, we get the **final user/item embeddings**  $\mathbf{h}_u^{(K)}$  and  $\mathbf{h}_v^{(K)}$ .
- For all  $u \in U, v \in V$ , we set
$$\mathbf{u} \leftarrow \mathbf{h}_u^{(K)}, \mathbf{v} \leftarrow \mathbf{h}_v^{(K)}.$$
- Score function is the inner product
$$\text{score}(u, v) = \mathbf{u}^T \mathbf{v}$$



# 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*.

GNN Applications: Recommender Systems

LightGCN



# LightGCN: Motivation (1)

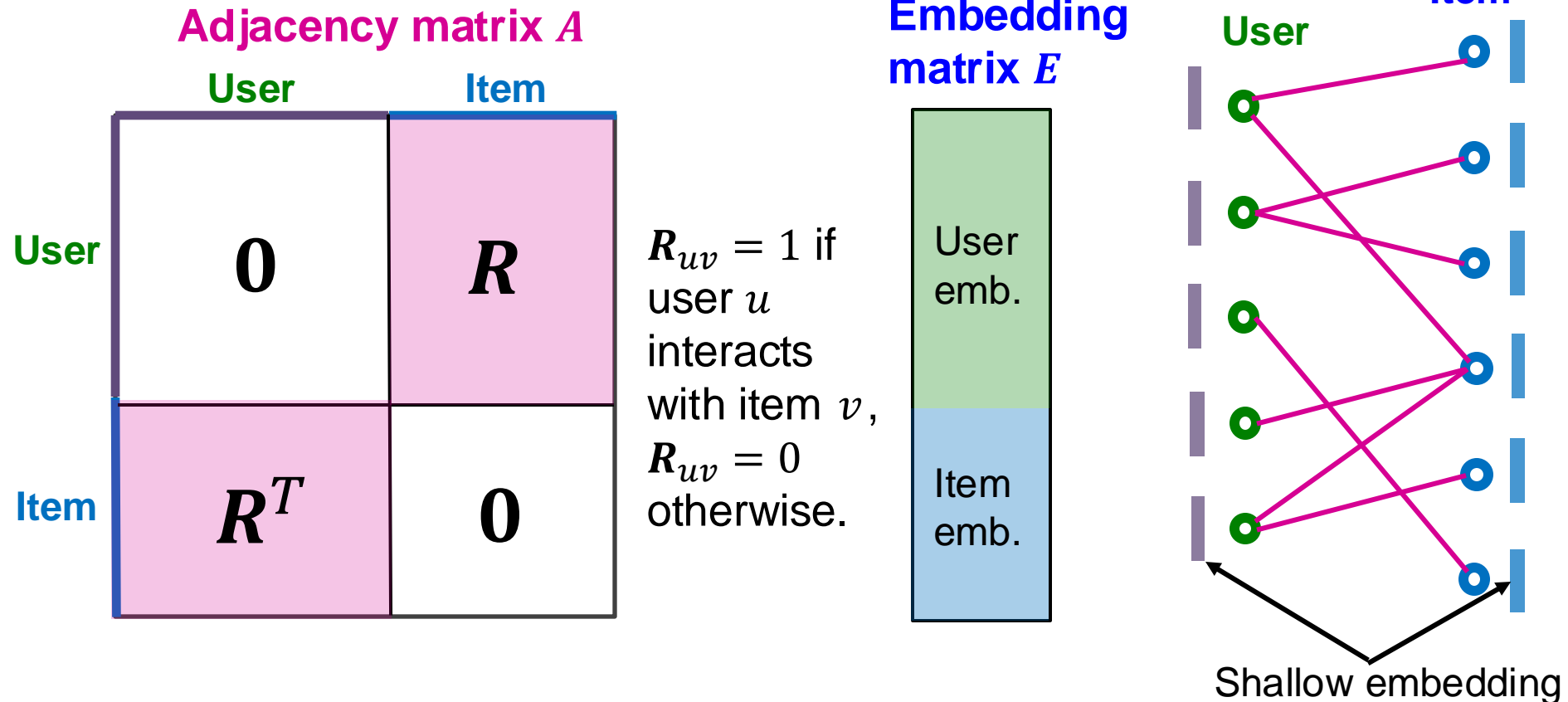
- **Recall:** NGCF jointly learns two kinds of parameters:
  - Shallow user/item embeddings
  - GNN's parameters
- **Observation:** Shallow learnable embeddings are already quite expressive.
  - They are learned for every (user/item) node.
  - Most of the parameter counts are in shallow embeddings when  $N$  (#nodes)  $\gg D$  (embedding dimensionality)
    - Shallow embeddings:  $O(ND)$ .
    - GNN:  $O(D^2)$ .
  - The GNN parameters may not be so essential for performance.

# LightGCN: Motivation (2)

- Can we simplify the GNN used in NGCF (e.g., remove its learnable parameters)?
  - **Answer:** Yes!
  - **Bonus:** Simplification improves the recommendation performance!
- **Overview of the idea:**
  - Adjacency matrix for a bipartite graph
  - Matrix formulation of GCN
  - Simplification of GCN by removing non-linearity
    - Related: SGC for scalable GNN [Wu et al. 2019]

# Recall: Adjacency and Embedding Matrices

- **Adjacency matrix** of a (undirected) bipartite graph.
- **Shallow embedding matrix.**



# Recall: Matrix Formulation of GCN

- **Recall:** The diffusion matrix of C&S.
- Let  $\mathbf{D}$  be the degree matrix of  $\mathbf{A}$ .
- Define the normalized adjacency matrix  $\tilde{\mathbf{A}}$  as
$$\tilde{\mathbf{A}} \equiv \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$$
- Let  $\mathbf{E}^{(k)}$  be the embedding matrix at  $k$ -th layer.
- Each layer of GCN's aggregation can be written in a matrix form:

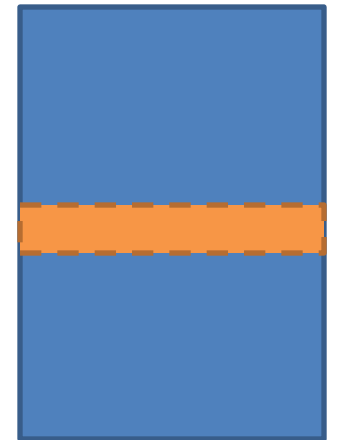
$$\mathbf{E}^{(k+1)} = \text{ReLU}(\tilde{\mathbf{A}} \mathbf{E}^{(k)} \mathbf{W}^{(k)})$$

Neighbor aggregation

Learnable linear transformation

**Note:** Different from the original GCN, self-connection is omitted here.

Matrix of node embeddings  $\mathbf{E}^{(k)}$



Each row stores node embedding

# Simplifying GCN (1)

- Simplify GCN by **removing ReLU non-linearity**:

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

Original idea from  
SGC [Wu et al. 2019]

- The final node embedding matrix is given as

$$\mathbf{E}^{(K)} = \tilde{\mathbf{A}} \mathbf{E}^{(K-1)} \mathbf{W}^{(K-1)}$$

$$\begin{aligned} &= \tilde{\mathbf{A}} (\tilde{\mathbf{A}} \mathbf{E}^{(K-2)} \mathbf{W}^{(K-2)}) \mathbf{W}^{(K-1)} \\ &= \tilde{\mathbf{A}} (\tilde{\mathbf{A}} (\dots (\tilde{\mathbf{A}} \mathbf{E}^{(0)} \mathbf{W}^{(0)}) \dots) \mathbf{W}^{(K-2)}) \mathbf{W}^{(K-1)} \\ &= \tilde{\mathbf{A}}^K \mathbf{E} (\mathbf{W}^{(0)} \dots \mathbf{W}^{(K-1)}) \end{aligned}$$

Set  $\mathbf{E}$  as input embedding  $\mathbf{E}^{(0)}$

# Simplifying GCN (2)

- Removing ReLU significantly simplifies GCN!

$$\mathbf{E}^{(K)} = \tilde{\mathbf{A}}^K \mathbf{E} \mathbf{W}$$

$$\mathbf{W} \equiv \mathbf{W}^{(0)} \dots \mathbf{W}^{(K-1)}$$

**Diffusing node embeddings  
along the graph**

(similar to C&S that diffuses soft  
labels along the graph)

- **Algorithm:** Apply  $\mathbf{E} \leftarrow \tilde{\mathbf{A}} \mathbf{E}$  for  $K$  times.
  - Each matrix multiplication diffuses the current embeddings to their one-hop neighbors.
  - **Note:**  $\tilde{\mathbf{A}}^K$  is dense and never gets materialized. Instead, the above iterative matrix-vector product is used to compute  $\tilde{\mathbf{A}}^K \mathbf{E}$ .

# Multi-Scale Diffusion

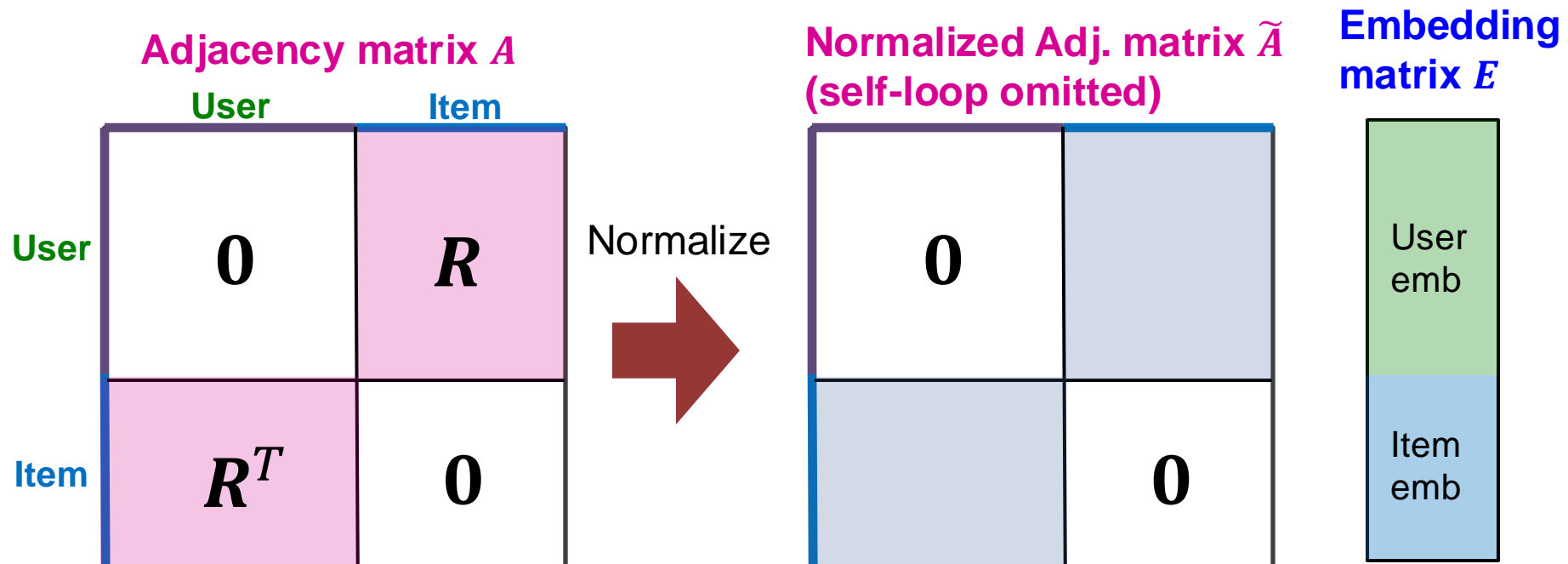
- We can consider **multi-scale diffusion**

$$\alpha_0 E^{(0)} + \alpha_1 E^{(1)} + \alpha_2 E^{(2)} + \dots + \alpha_K E^{(K)}$$

- The above includes embeddings diffused at multiple hop scales.
- $\alpha_0 E^{(0)} = \alpha_0 \tilde{A}^0 E^{(0)}$  acts as a self-connection (that is omitted in the definition  $\tilde{A}$ )
- The coefficients,  $\alpha_0, \dots, \alpha_K$ , are hyper-parameters.
- For simplicity, LightGCN uses the uniform coefficient, i.e.,  $\alpha_k = \frac{1}{K+1}$  for  $k = 0, \dots, K$ .

# LightGCN: Model Overview (1)

- **Given:**
  - Adjacency matrix  $A$
  - Initial learnable embedding matrix  $E$



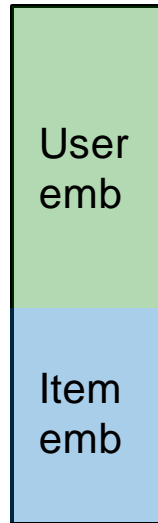


# LightGCN: Model Overview (2)

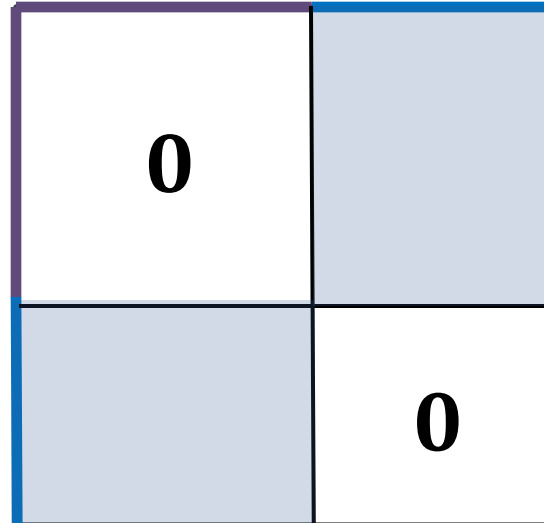
- Iteratively diffuse embedding matrix  $E$  using  $\tilde{A}$

For  $k = 0 \dots K - 1$ ,

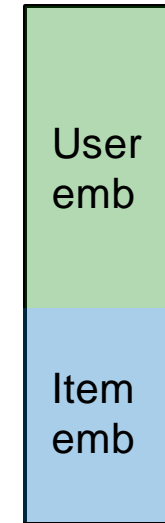
Embedding  
matrix  $E^{(k+1)}$



Normalized Adj. matrix  $\tilde{A}$   
(self-loop omitted)

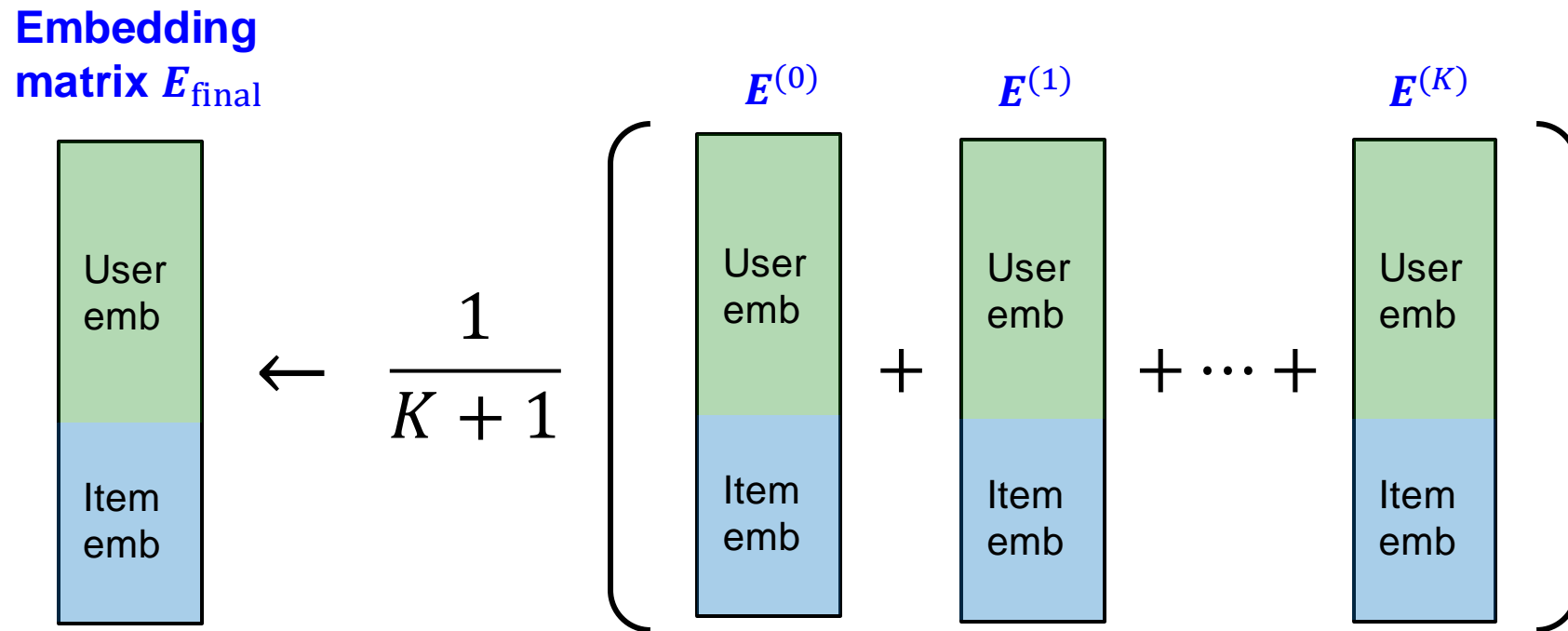


Embedding  
matrix  $E^{(k)}$   
( $E^{(0)}$  is set to  $E$ )



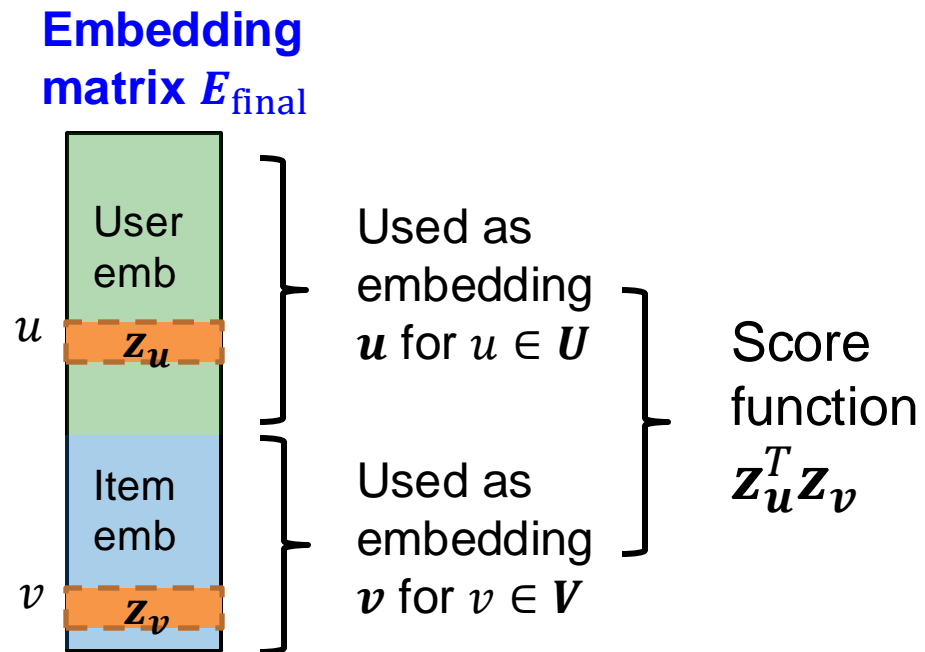
# LightGCN: Model Overview (3)

- Average the embedding matrices at different scales.



# LightGCN: Model Overview (4)

- **Score function:**
  - Use user/item vectors from  $E_{\text{final}}$  to score user-item interaction



# LightGCN: Summary

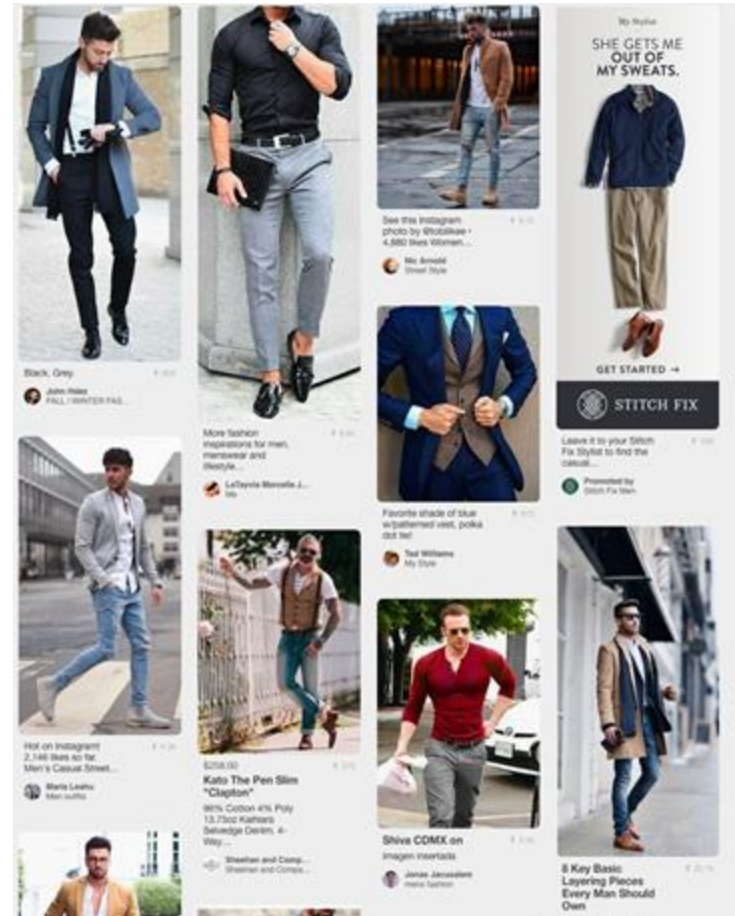
- LightGCN simplifies NGCF by **removing the learnable parameters of GNNs.**
- **Learnable parameters are all in the shallow input node embeddings.**
  - Diffusion propagation only involves matrix-vector multiplication.
  - Key insight: the diffusion directly encourages the **embeddings of similar users/items to be similar**
  - The simplification leads to better empirical performance than NGCF.

# GNN Applications: Recommender Systems

## PinSAGE

# Motivation

- P2P recommendation



# PinSAGE: Pin Embedding

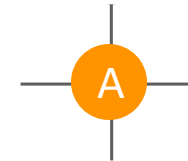
- Unifies visual, textual, and graph information.
- The largest industry deployment of a Graph Convolutional Networks.
- Huge Adoption across Pinterest
- Works for fresh content and is available in a few seconds after pin creation



# Application: Pinterest

## PinSage graph convolutional network:

- **Goal:** Generate embeddings for nodes in a large-scale Pinterest graph containing billions of objects
- **Key Idea:** Borrow information from nearby nodes
  - E.g., bed rail Pin might look like a garden fence, but gates and beds are rarely adjacent in the graph



- Pin embeddings are essential to various tasks like recommendation of Pins, classification, ranking
  - Services like “Related Pins”, “Search”, “Shopping”, “Ads”



# Harnessing Pins and Boards



**Very ape blue structured coat**

Nitty Gritty



Picked for you  
Street style

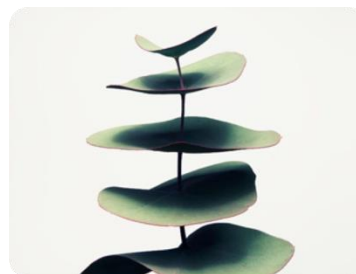


**Hans Wegner chair**

Room and Board



Promoted by  
Room & Board

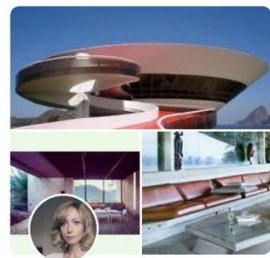


This is just a beautiful image for thoughts.  
Yay or nay, your choice.

↑ 14



Annie Teng  
Plantation



**mid century modern ...**  
MJLI -



**Man Style**  
Gavin Jones



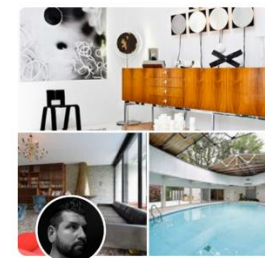
**men + style |**  
FIG + SALT



**Plants**  
HelloSandwich



**Men's Style**  
Andrea Sempi



**Mid century modern**  
Tyler Goodro



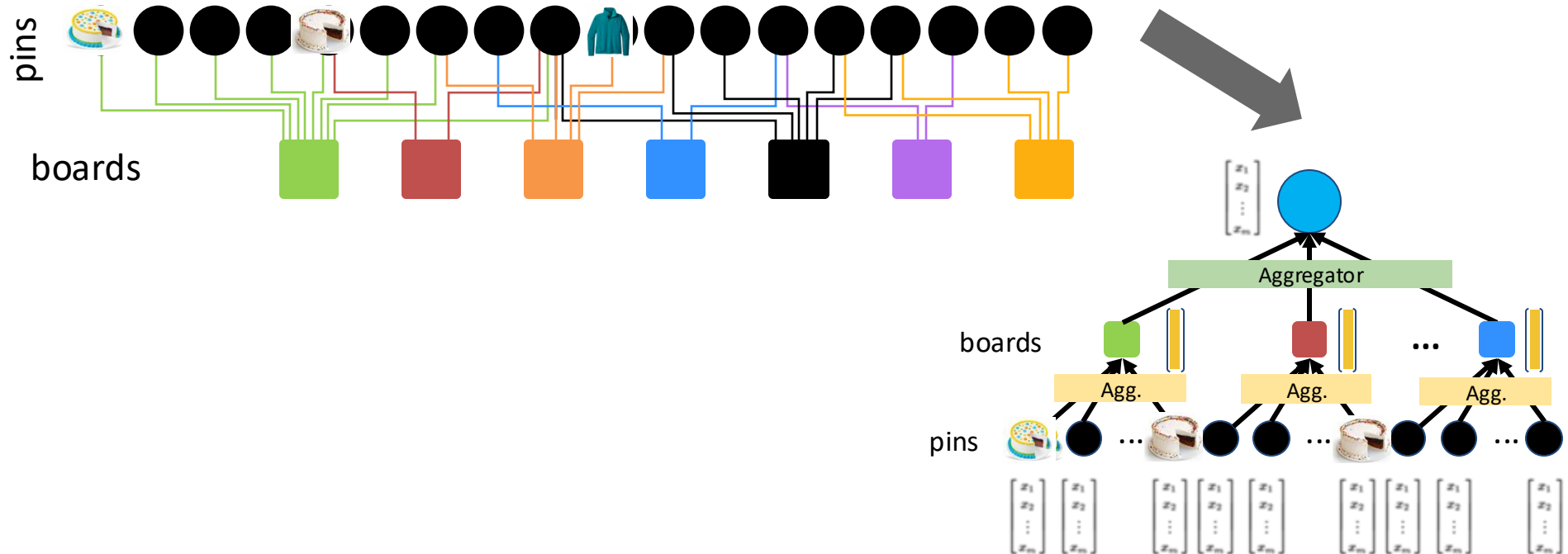
**Plants**  
Moorea Seal



**Mid century modern ...**  
Prettygreentea

# PinSAGE: Graph Neural Network

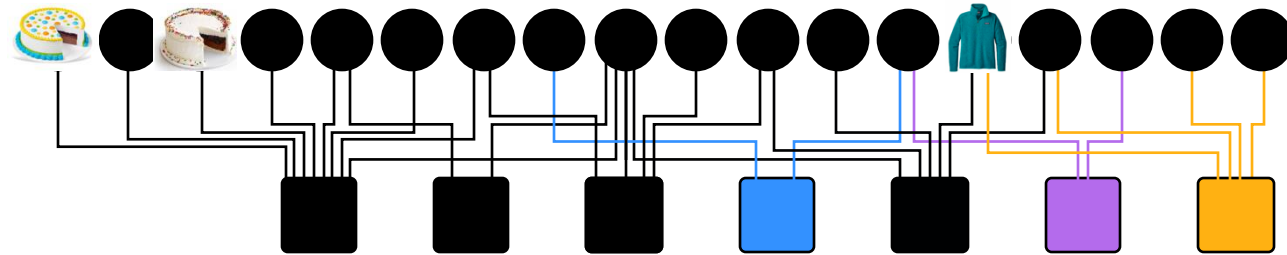
- Graph has tens of billions of nodes and edges
- Further resolves embeddings across the Pinterest graph



# PinSAGE: Scaling Up and Considerations

- In addition to the GNN model, the PinSAGE paper introduces several methods to scale the GNN to a billion-scale recommender system (e.g., Pinterest).
  - **Mini-batch** training of GNNs on a large-graph
  - **Shared negative samples across users in a mini-batch**
  - Hard negative samples
  - Curriculum learning

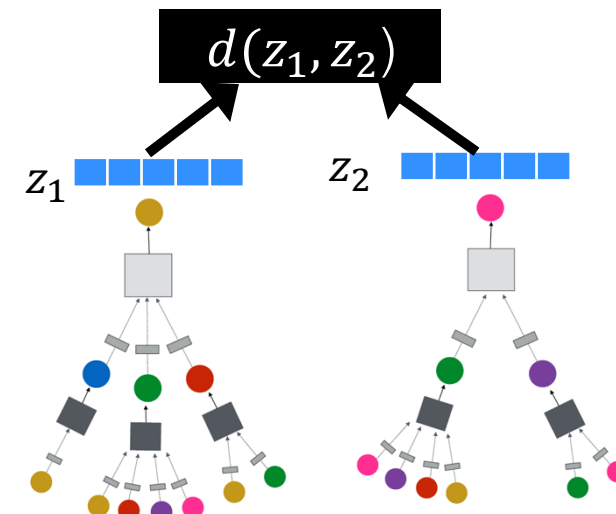
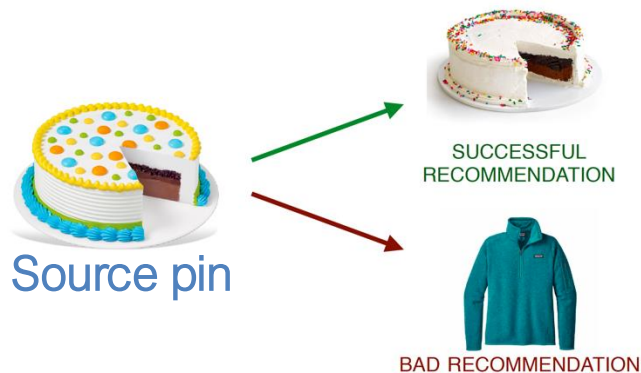
# PinSAGE Model



- **Task:** Recommend related pins to users

- Learn node embeddings  $z_i$  such that

- $d(z_{cake1}, z_{cake2}) < d(z_{cake1}, z_{sweater})$



# Training Data

1+B repin pairs:

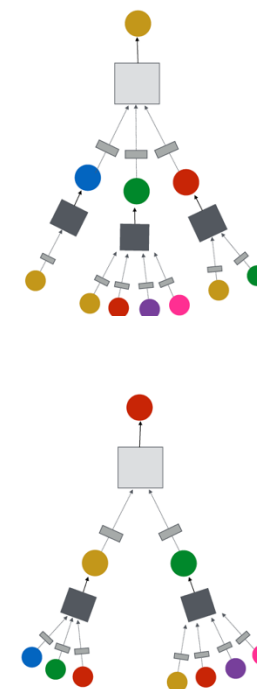
- From Related Pins surface
- Capture semantic relatedness
- Goal: Embed such pairs to be “neighbors”

Example positive training pairs (Q,X):



# Shared Negative Samples (1)

- **Recall:** In BPR loss, for each user  $u^* \in \mathbf{U}_{\text{mini}}$ , we sample one positive item  $v_{\text{pos}}$  and a set of sampled negative items  $\mathbf{V}_{\text{neg}} = \{v_{\text{neg}}\}$ .
- Using more negative samples per user improves the recommendation performance, but is also expensive.
  - We need to generate  $|\mathbf{U}_{\text{mini}}| \cdot |\mathbf{V}_{\text{neg}}|$  embeddings for negative nodes.
  - We need to apply  $|\mathbf{U}_{\text{mini}}| \cdot |\mathbf{V}_{\text{neg}}|$  GNN computational graphs (see right), which is expensive.



# Shared Negative Samples (2)

- **Key idea:** We can share the same set of negative samples  $V_{\text{neg}} = \{v_{\text{neg}}\}$  *across all users*  $U_{\text{mini}}$  in the mini-batch.
- This way, we only need to generate  $|V_{\text{neg}}|$  embeddings for negative nodes.
  - This saves the node embedding generation computation **by a factor of  $|U_{\text{mini}}|$** !
  - Empirically, the performance stays similar to the non-shared negative sampling scheme.

# Hard Negatives (1)

- **Challenge:** Industrial recsys needs to make **extremely fine-grained predictions**.
  - #Total items: Up to billions.
  - #Items to recommend for each user: 10 to 100.
- **Issue:** The shared negative items are randomly sampled from all items
  - Most of them are “**easy negatives**”, i.e., a model does not need to be fine-grained to distinguish them from positive items.
- We need a way to sample “**hard negatives**” to force the model to be fine-grained!



# PinSAGE: Curriculum Learning

- **Idea:** use harder and harder negative samples
- Include more and more hard negative samples for each epoch



Source pin



Positive



Easy negative



Hard negative

# Curriculum Learning

- **Key insight:** It is effective **to make the negative samples *gradually harder* in the process of training.**
- At  $n$ -th epoch, we add  $n - 1$  hard negative items.
  - #(Hard negatives) gradually increases in the process of training.
- The model will gradually learn to make finer-grained predictions.

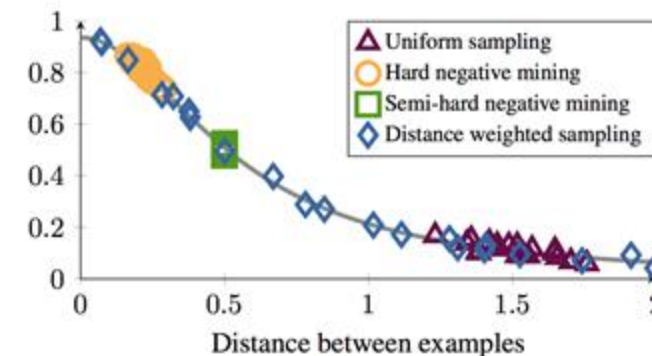
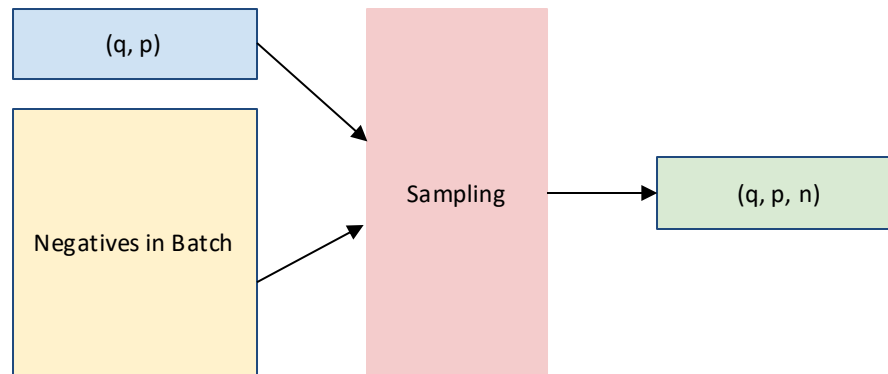
# Hard Negatives (2)

- For each user node, the **hard negatives** are item nodes that are close (but not connected) to the user node in the graph.
- Hard negatives for user  $u \in U$  are obtained as follows:
  - Compute **personalized page rank** (PPR) for user  $u$ .
  - Sort items in the descending order of their PPR scores.
  - Randomly sample item nodes that are ranked high but not too high, e.g., 2000<sup>th</sup> — 5000<sup>th</sup>.
    - Item nodes that are close but not too close (connected) to the user node.
- The hard negatives for each user are used in addition to the shared negatives.

# PinSAGE: Negative Sampling

$(q, p)$  positive pairs are given but various methods to sample negatives to form  $(q, p, n)$

- Distance Weighted Sampling ([Wu et al., 2017](#))
  - Sample negatives so that query-negative distance distribution is approx  $U[0.5, 1.4]$



(b) Sample distribution for different strategies.

# Fine-Grained Object Similarity

Query



Visual only

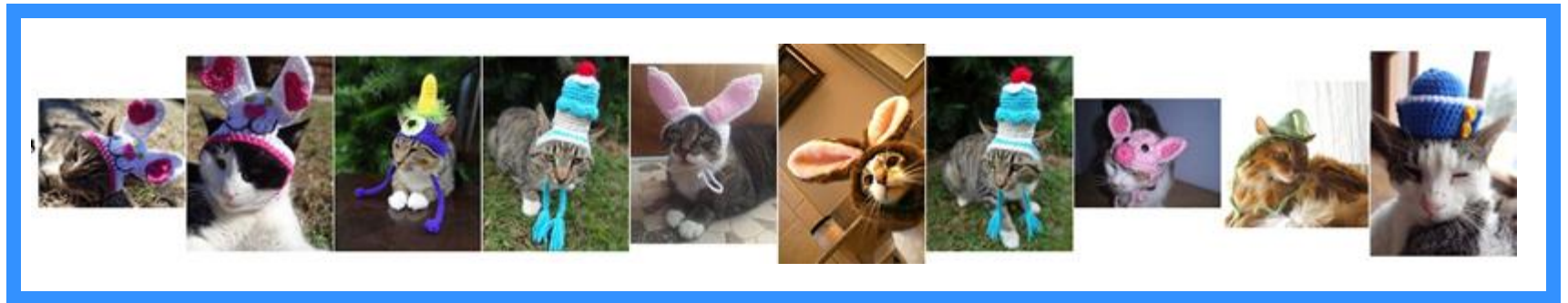


PinSAGE



# Compare against Production Model

Query



# PinSAGE: Summary

- **PinSAGE uses GNNs** to generate high-quality user/item embeddings that **capture both the rich node attributes and graph structure**.
- The PinSAGE model is effectively trained using sophisticated **negative sampling strategies**.
- PinSAGE is **successfully deployed at Pinterest**, a billion-scale image content recommendation service.
  - **Uncovered in this lecture**: How to **scale up GNNs to large-scale graphs**. Will be covered in a later lecture.