# COMP4222 Machine Learning with Structured Data

Recommender Systems 2

Instructor: Yangqiu Song

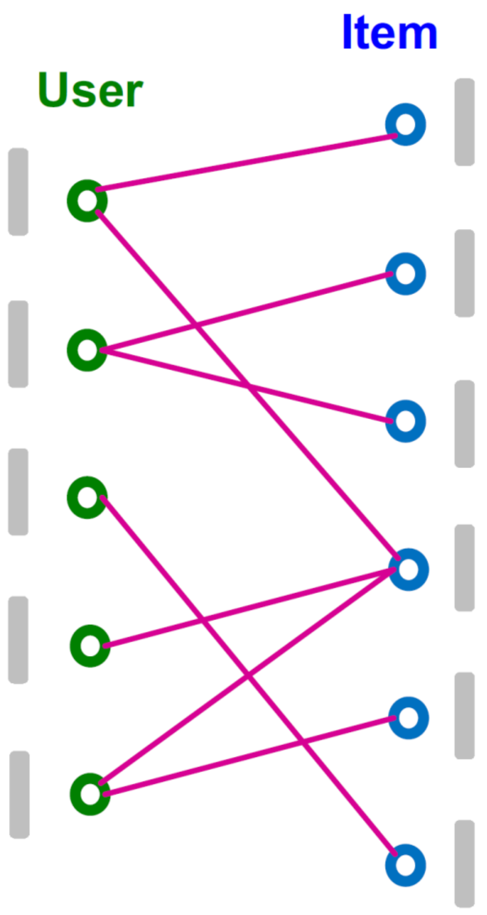
**Slides credits: Jure Laskovec**

## Presentation

- Starts from next week
- Rubrics
  - Content Clearness (20%)
  - Description of Methods (20%)
  - Experiments and Components (20%)
  - Insight Sharing (20%)
  - Time Management (20%)
- Each group will be graded by me and 2 Tas
  - Average will be taken as the final score

# Conventional Collaborative Filtering

- Conventional collaborative filtering model is based on **Matrix Factorization (MF)**.
  - Use shallow encoders for users and items
    - For every $u \in \boldsymbol{U}$ and $v \in \boldsymbol{V}$, we prepare shallow learnable embeddings $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{\mathrm{D}}$.
  - Score function for user $u$ and item $v$ is $f(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^{\mathrm{D}}\boldsymbol{v}$.



Learnable shallow user/item embeddings

## Limitations of MF

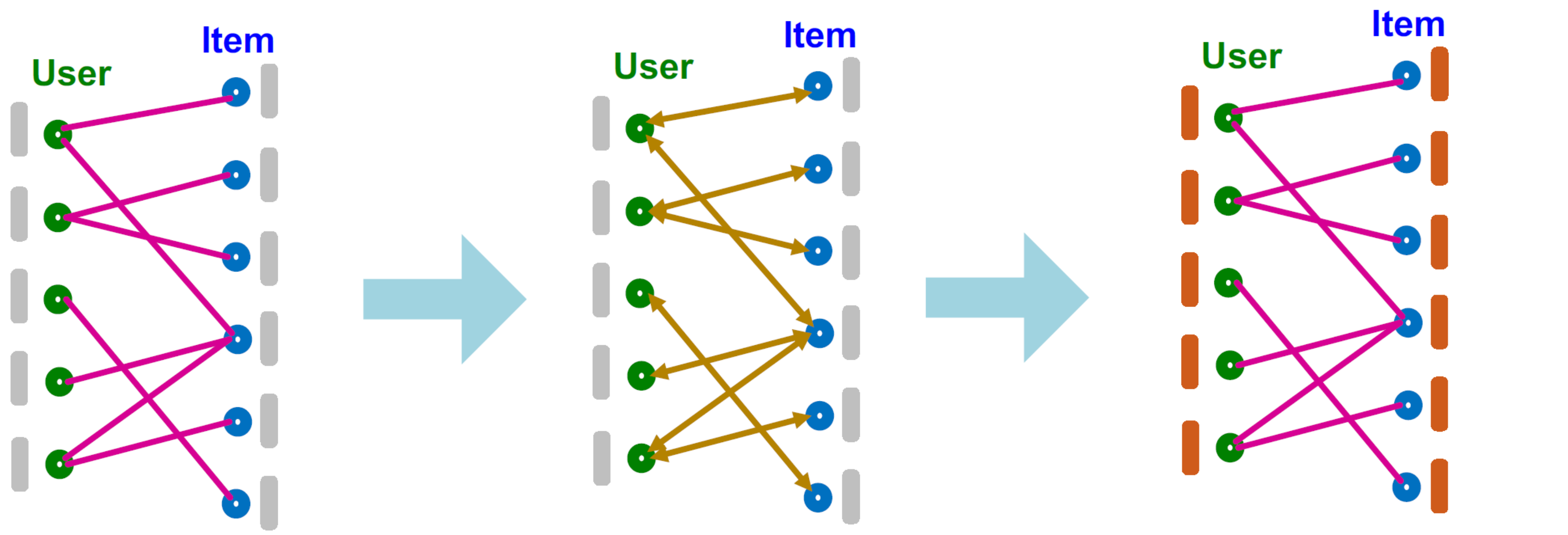
- The model itself does ***not explicitly*** capture graph structure
    - The graph structure is ***only implicitly*** captured in the training objective.
- Only the **first-order graph structure** (i.e., edges) is captured in the training objective.
    - **High-order graph structure** (e.g., $K$-hop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



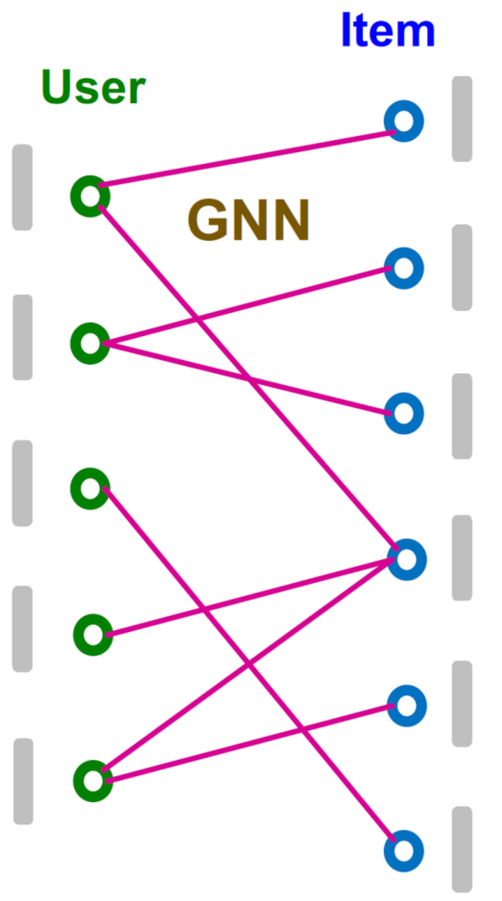
Initial shallow embeddings (not graph-aware)

Use a GNN to propagate embeddings

NGCF's graph-aware embeddings

# NGCF Framework

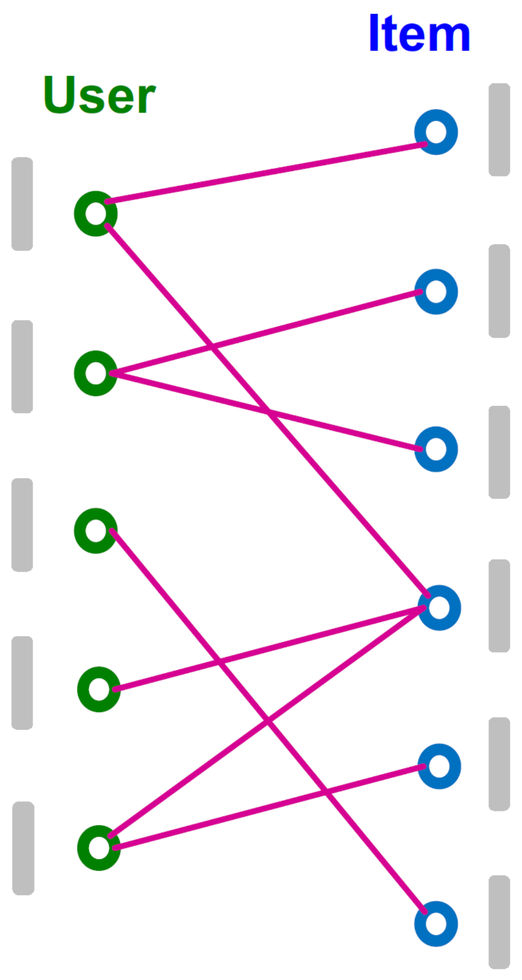
- **Given**: User-item bipartite graph.
- **NGCF framework:**
  - Prepare shallow learnable embedding for each node.
  - Use multi-layer GNNs to propagate embeddings along the bipartite graph.
    - High-order graph structure is captured.
  - Final embeddings are *explicitly* graph- aware!
- **Two kinds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



Shallow user/item embeddings (learnable)

# Initial Node Embeddings

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



Learnable shallow user/item embeddings

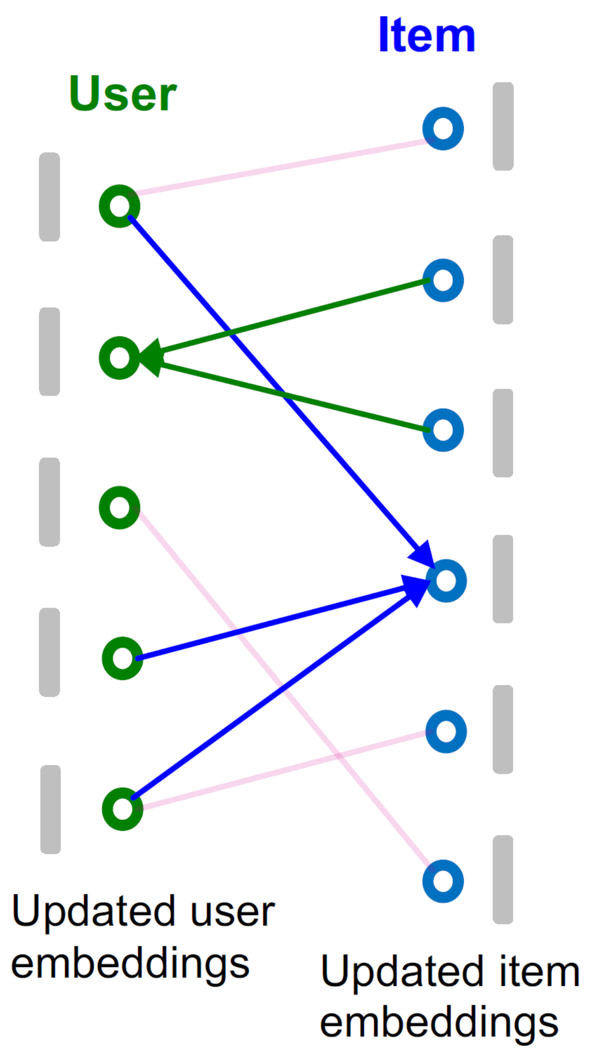
# Neighbor Aggregation

- Iteratively update node embeddings using neighboring embeddings.

$$h_v^{(k+1)} = \mathrm{COMBINE}\left(h_v^{(k)}, \mathrm{AGGR}\left(\left\{h_u^{(k)}\right\}_{u\in N(v)}\right)\right)$$

$$h_u^{(k+1)} = \mathrm{COMBINE}\left(h_u^{(k)}, \mathrm{AGGR}\left(\left\{h_v^{(k)}\right\}_{v\in N(u)}\right)\right)$$

- **High-order graph structure is captured through iterative neighbor aggregation.**
- Different architecture choices are possible for $\mathrm{AGGR}$ and $\mathrm{COMBINE}$.
  - $\mathrm{AGGR}(\cdot)$ can be $\mathrm{MEAN}\ \cdot$
  - $\mathrm{COMBINE}(\boldsymbol{x}, \boldsymbol{y})$ can be $\mathrm{ReLU}\ \mathrm{Linear}(\mathrm{Concat}(\boldsymbol{x}, \boldsymbol{y}))$



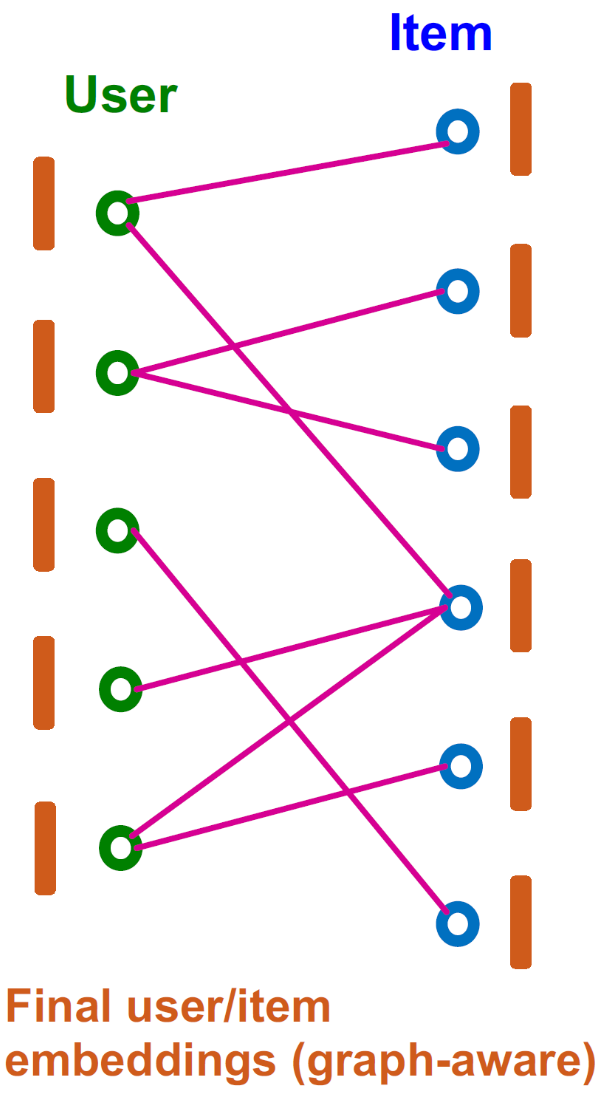
# Final Embeddings and Score Function

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

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

- Score function is the inner product:

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



## 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.**

## LightGCN

# LightGCN: Motivation

- **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 node.
  - Most of the parameter counts are in shallow embeddings when $N$ (#nodes) $\gg$ $D$ (embedding dimensionality)
    - Shallow embeddings: $O(ND)$.
    - GNN: $O(D^2)$.

## LightGCN: Motivation

- 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]

# Adjacency and Embeddings Matrices

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

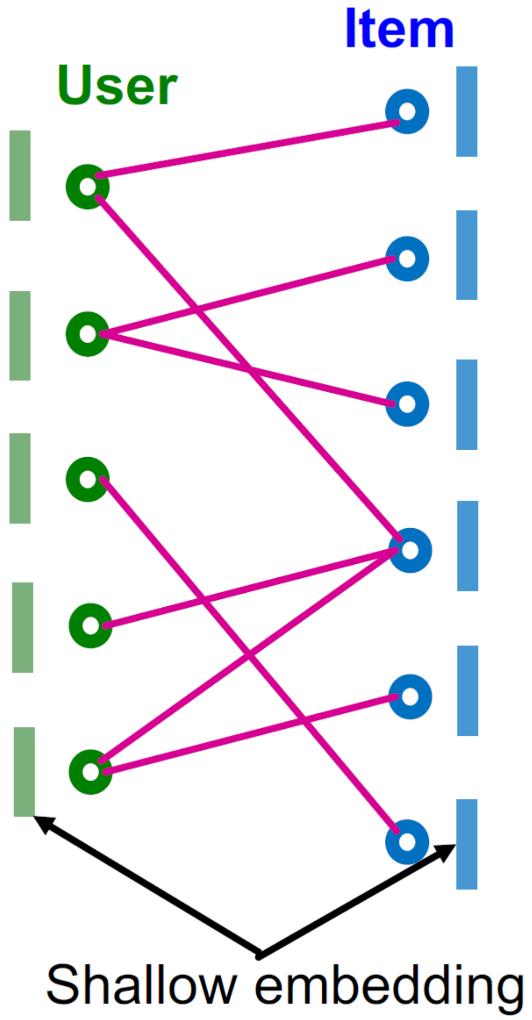
**Adjacency matrix $A$**

$$A = \begin{array}{c|cc} & \text{User} & \text{Item} \\ \hline \text{User} & \mathbf{0} & \boldsymbol{R} \\ \text{Item} & \boldsymbol{R}^T & \mathbf{0} \end{array}$$

$\boldsymbol{R}_{uv} = 1$ if user $u$ interacts with item $v$, $\boldsymbol{R}_{uv} = 0$ otherwise.

**Embedding matrix $\boldsymbol{E}$**

| |
|---|
| User emb. |
| Item emb. |



# Matrix Formulation of GCN

- Let $\boldsymbol{D}$ be the degree matrix of $\boldsymbol{A}$.
- Define the normalized adjacency matrix $\widetilde{\boldsymbol{A}}$ as

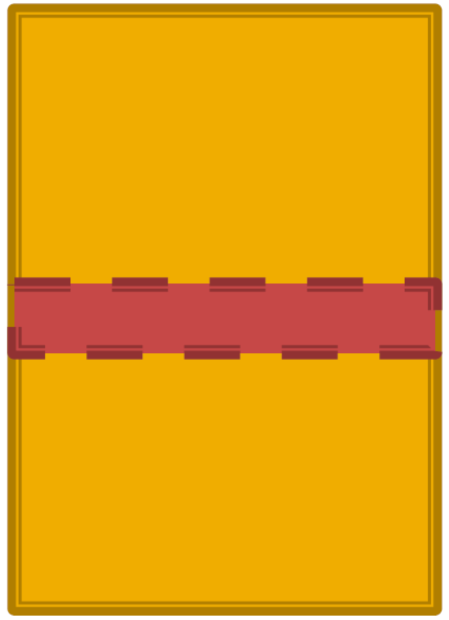
$$\widetilde{\boldsymbol{A}} \equiv \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$$

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

- Let $\boldsymbol{E}^{(k)}$ be the embedding matrix at $k$-th layer.
- Each layer of GCN's aggregation can be written in a matrix form:

$$\boldsymbol{E}^{(k+1)} = \mathrm{ReLU}(\underbrace{\boxed{\widetilde{\boldsymbol{A}}\boldsymbol{E}^{(k)}}}_{\text{Neighbor aggregation}}\underbrace{\boxed{\boldsymbol{W}^{(k)}}}_{\text{Learnable linear transformation}})$$

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



Each row stores node embedding

Original idea from SGC [Wu et al. 2019]
https://arxiv.org/abs/1902.07153

## Simplifying GCN

- **Simplify GCN by removing ReLU non-linearity:** $\boldsymbol{E}^{(k+1)} = \widetilde{\boldsymbol{A}}\boldsymbol{E}^{(k)}\boldsymbol{W}^{(k)}$
- The final node embedding matrix is given as

$$\boldsymbol{E}^{(K)} = \widetilde{\boldsymbol{A}}\ \underbrace{\boldsymbol{E}^{(K-1)}}\boldsymbol{W}^{(K-1)}$$

$$= \widetilde{\boldsymbol{A}}\left(\widetilde{\boldsymbol{A}}\underbrace{\boldsymbol{E}^{(K-2)}}\boldsymbol{W}^{(K-2)}\right)\boldsymbol{W}^{(K-1)}$$

$$= \underbrace{\widetilde{\boldsymbol{A}}\left(\widetilde{\boldsymbol{A}}\left(\cdots\left(\widetilde{\boldsymbol{A}}\right.\right.\right.}\underbrace{\boldsymbol{E}^{(0)}}\underbrace{\left.\left.\left.\boldsymbol{W}^{(0)}\right)\cdots\right)\boldsymbol{W}^{(K-2)}\right)\boldsymbol{W}^{(K-1)}}$$

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

$$= \widetilde{\boldsymbol{A}}^{K}\ \boldsymbol{E}\ \left(\boldsymbol{W}^{(0)}\cdots\boldsymbol{W}^{(K-1)}\right)$$

# Simplifying GCN

- Removing ReLU significantly simplifies GCN!

$$\boldsymbol{E}^{(K)} = \boxed{\widetilde{\boldsymbol{A}}^{K}\ \boldsymbol{E}}\,\boldsymbol{W} \qquad \boldsymbol{W} \equiv \boldsymbol{W}^{(0)} \dots \boldsymbol{W}^{(K-1)}$$

- **It's considered as diffusing node embeddings along the graph**

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

## Multi-scale Diffusion

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

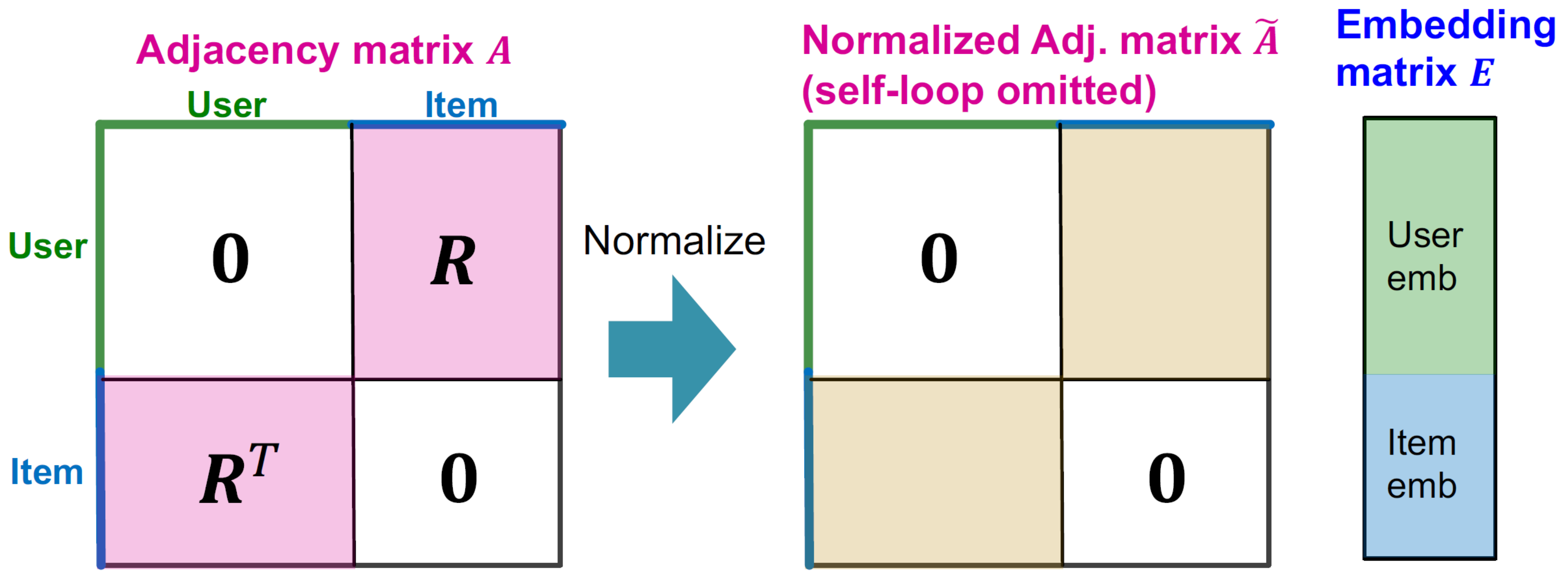
$$\alpha_0 E^{(0)} + \alpha_1 E^{(1)} + \alpha_2 E^{(2)} + \cdots + \alpha_K E^{(K)}$$

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

$$\alpha_k = \frac{1}{K+1} \text{ for } k = 0, \dots, K.$$

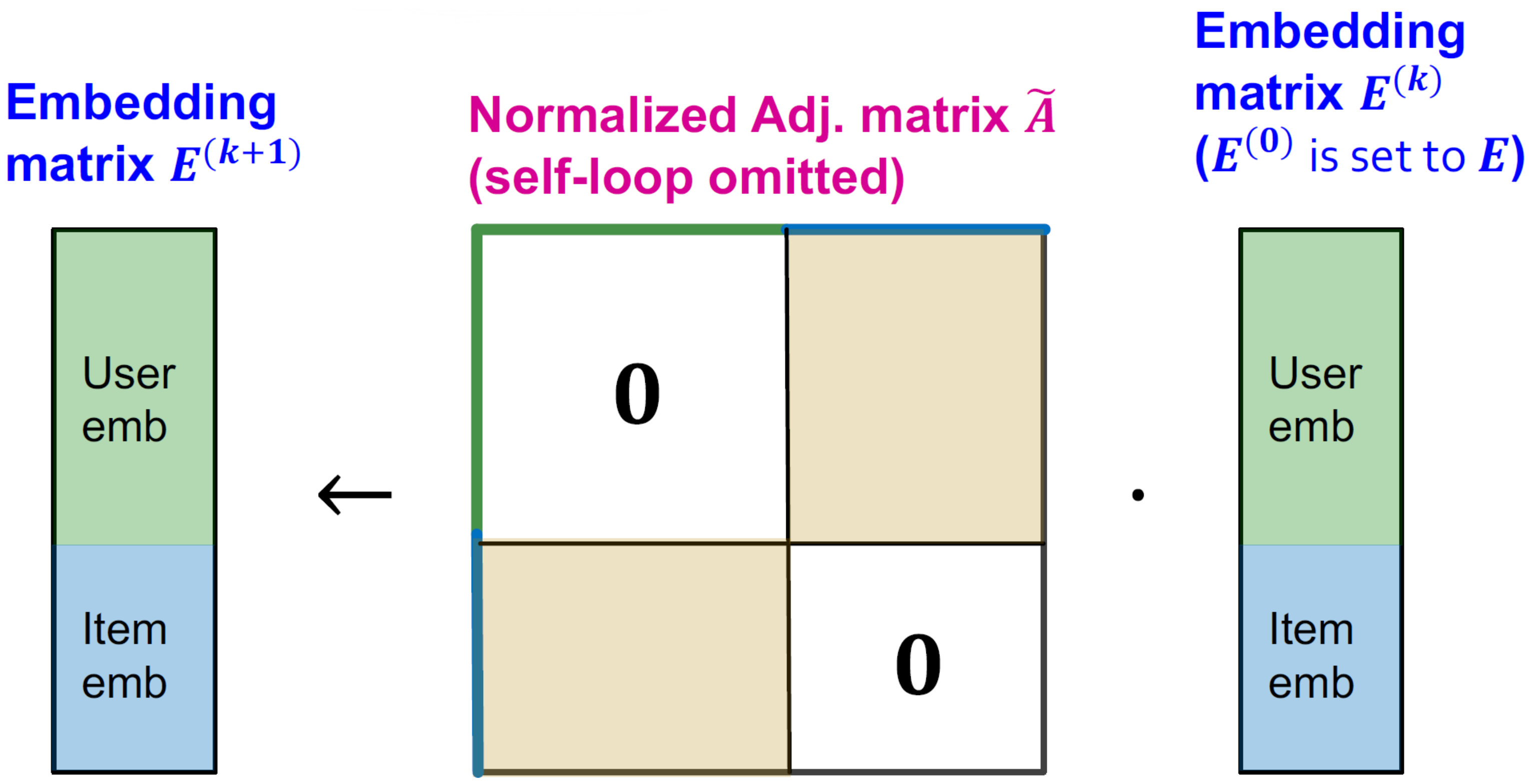
# LightGCN: Model Overview

- **Given**:
  - **Adjacency matrix A**
  - **Initial learnable embedding matrix $\boldsymbol{E}$**



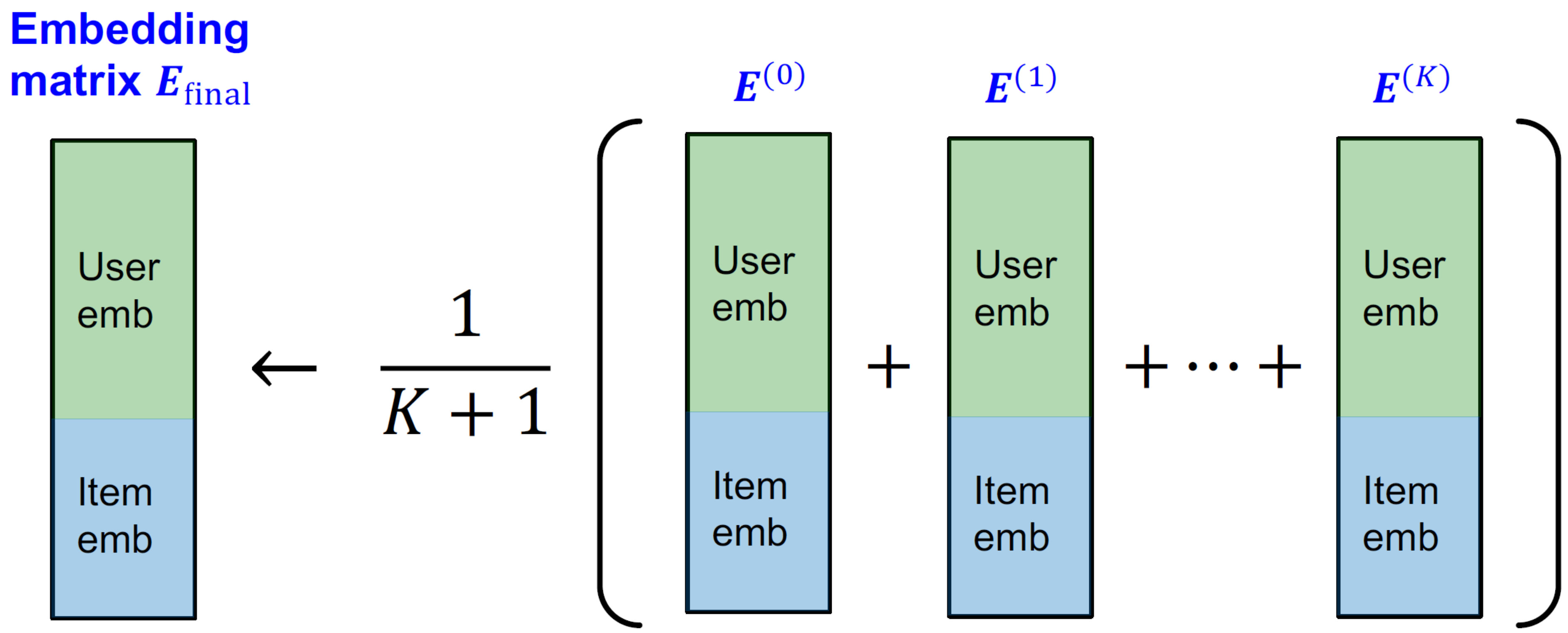
# LightGCN: Model Overview

- Iteratively diffuse embedding matrix $\boldsymbol{E}$ using $\widetilde{\boldsymbol{A}}$
- For $k = 0 \dots K - 1,$


Embedding matrix $E^{(k+1)}$
User emb
Item emb
$\leftarrow$
Normalized Adj. matrix $\widetilde{A}$ (self-loop omitted)
$\mathbf{0}$
$\mathbf{0}$
$\cdot$
Embedding matrix $E^{(k)}$ ($E^{(0)}$ is set to $E$)
User emb
Item emb

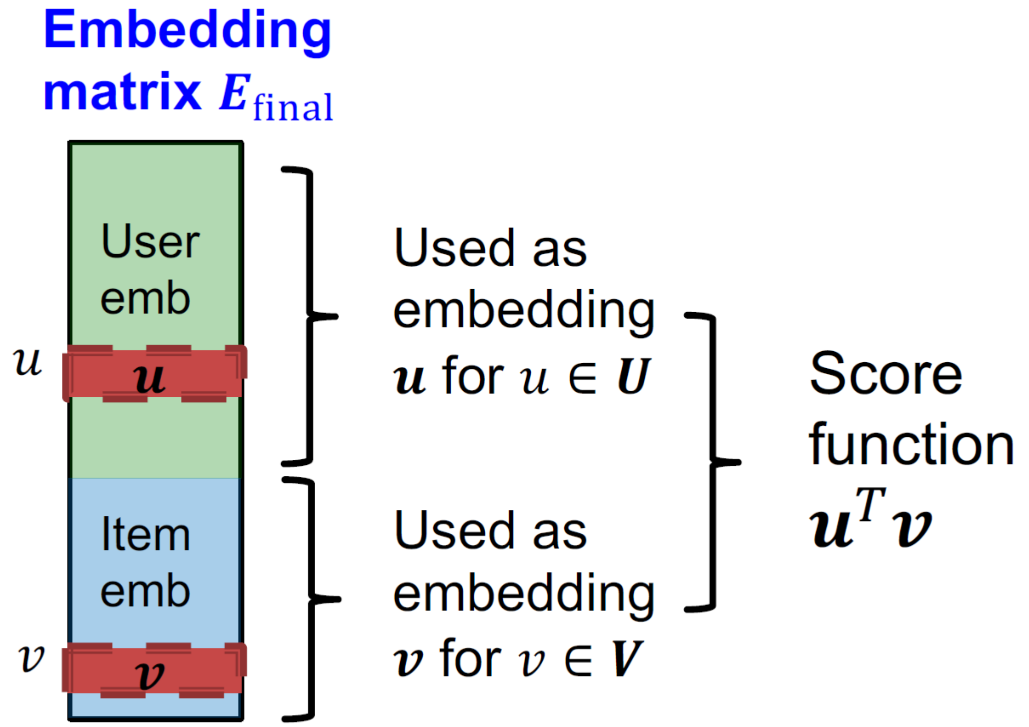
# LightGCN: Model Overview

- Average the embedding matrices at  different scales.



# LightGCN: Model Overview

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



Embedding matrix $\boldsymbol{E}_{\text{final}}$
User emb
$u$
$\boldsymbol{u}$
Item emb
$v$
$\boldsymbol{v}$
Used as embedding $\boldsymbol{u}$ for $u \in \boldsymbol{U}$
Used as embedding $\boldsymbol{v}$ for $v \in \boldsymbol{V}$
Score function $\boldsymbol{u}^T\boldsymbol{v}$

## LightGCN: Intuition

- **Question**: **Why does the simple diffusion propagation work well?**
- **Answer**: The diffusion directly encourages the embeddings of similar users/items to be similar.
  - Similar users share many common neighbors (items) and are expected to have similar future preferences (interact with similar items).

- Remember $(I - \alpha D^{-1}W)^{-1} = \lim_{t \to \infty} \sum_{k=0}^{t-1} (\alpha D^{-1}W)^k$
- If $\boldsymbol{L} = \boldsymbol{U\Lambda U^T}$, then $\boldsymbol{L^{-1}} = \boldsymbol{U\Lambda^{-1}U^T}$
  - $\boldsymbol{L^{-1}}$ is considered as a low-pass filter

## LightGCN and MF: Comparison

- Both LightGCN and Matrix Factorization (MF) **learn a unique embedding for each user/item.**
- The difference is that
  - MF directly uses the shallow user/item embeddings  for scoring.
  - LightGCN uses the ***diffused*** user/item embeddings  for scoring.
- LightGCN performs better than MF but are also more computationally expensive due to the  additional diffusion step.
  - The final embedding of a user/item is obtained by aggregating embeddings of its multi-hop neighboring nodes.

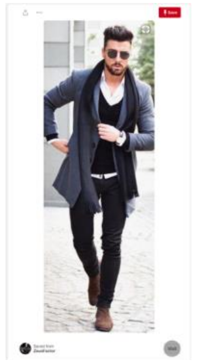
# LightGCN: Summary

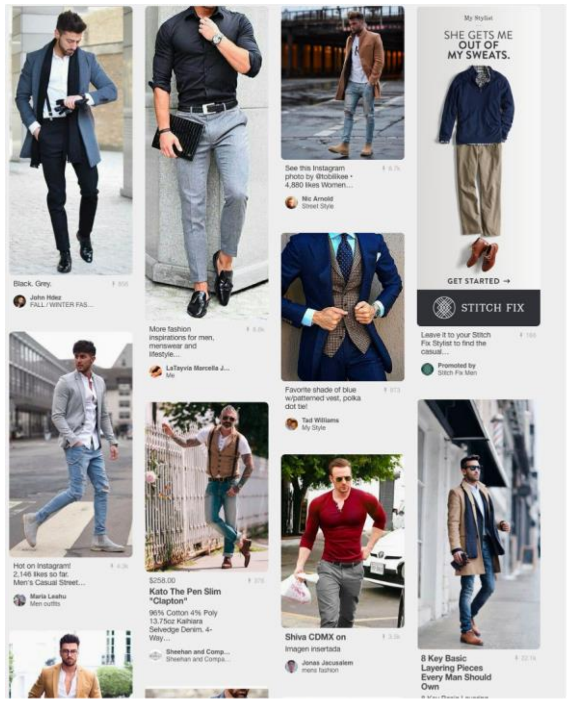
- LightGCN simplifies NGCF by **removing the learnable parameters of GNNs.**
- **Learnable parameters are all in the shallow input node embeddings.**
  - Diffusion propagation only involves matrix-vector multiplication.
  - The simplification leads to better empirical performance than NGCF.

## 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

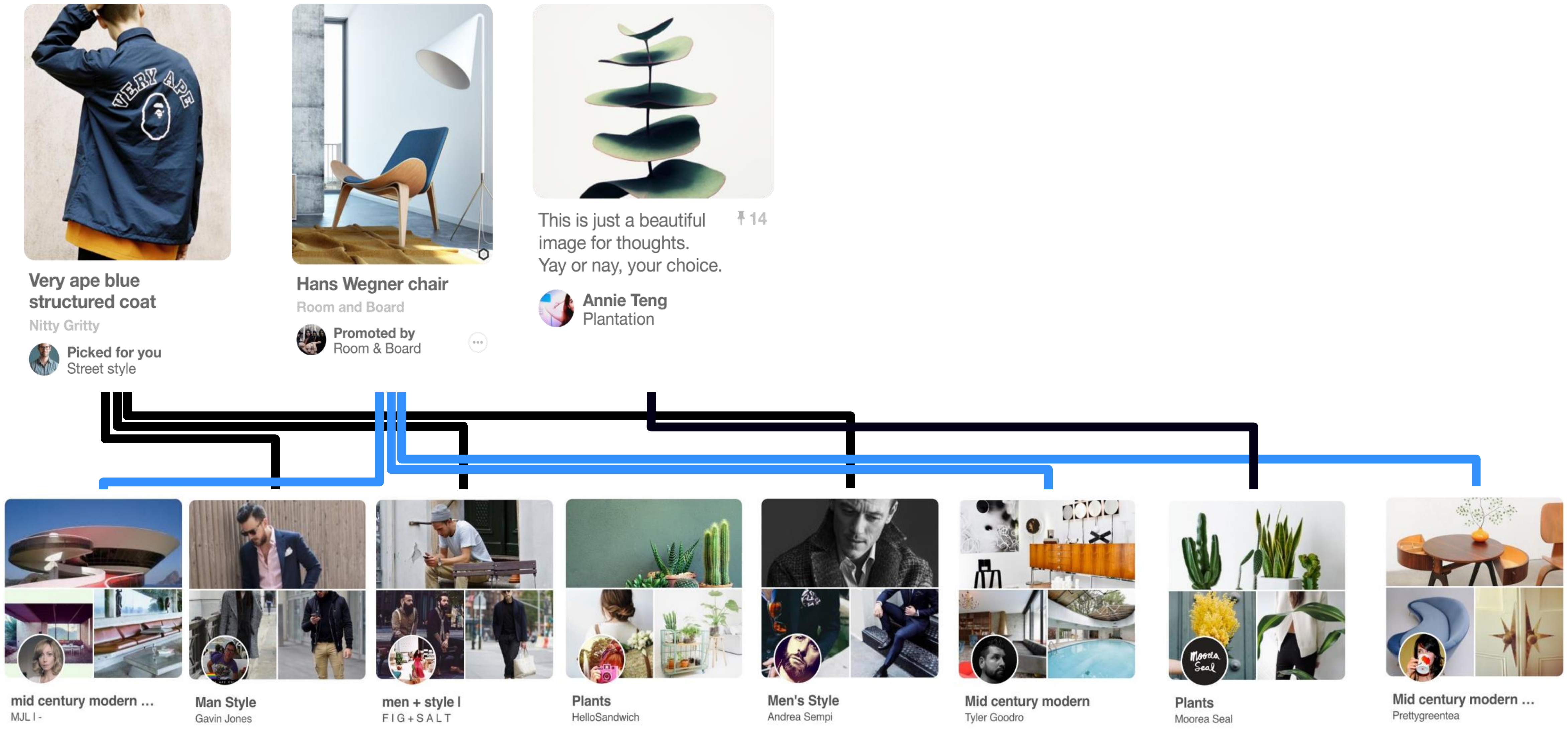
Graph Convolutional Neural Networks for Web-Scale Recommender Systems, Ying et al., 2018

# 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

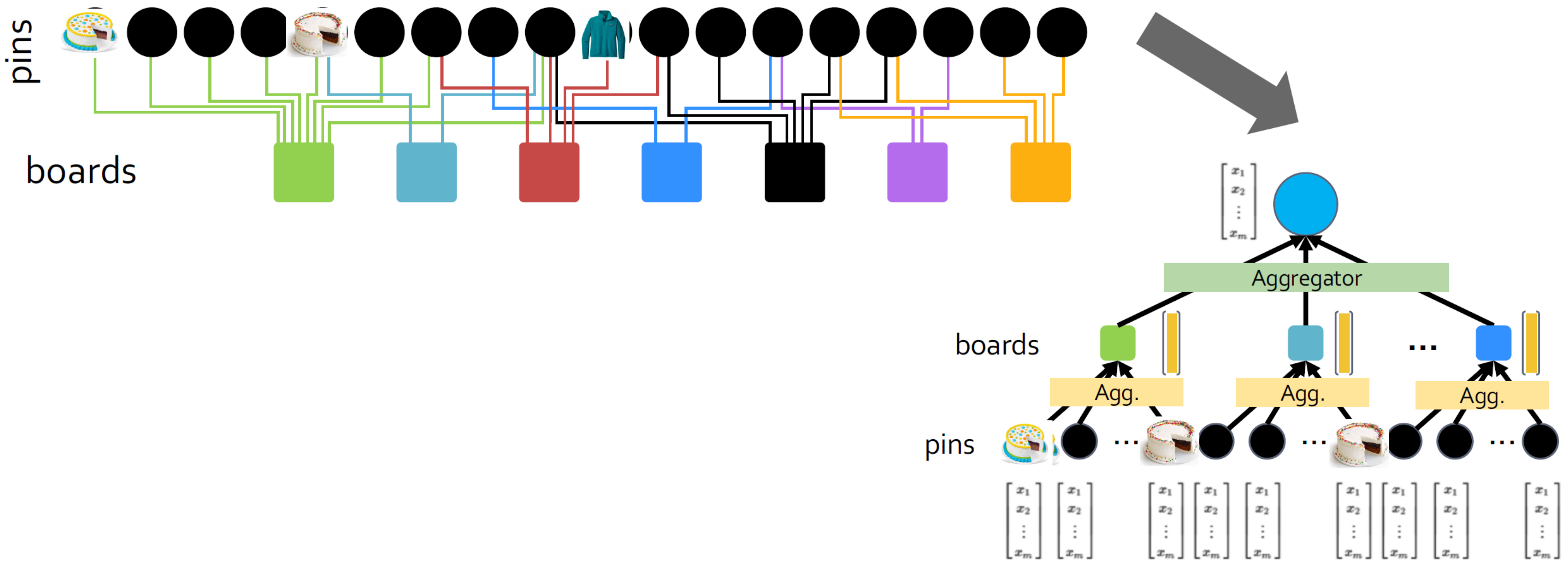
  - Pin embeddings are essential to various tasks like recommendation of Pins, classification, ranking
    - Services like “Related Pins”, “Search”, “Shopping”, “Ads”

# Harnessing Pins and Boards



# PinSAGE: Graph Neural Network

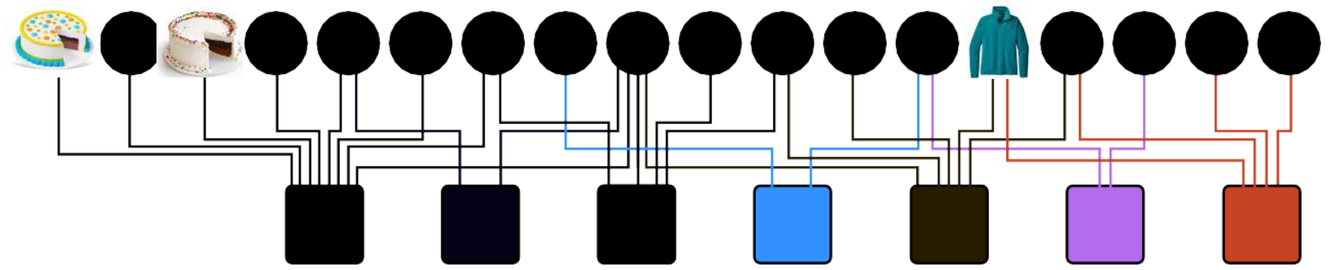
- Graph has tens of billions of nodes and edges
- Further resolves embeddings across the  Pinterest graph



# PinSAGE: Methods for Scaling Up

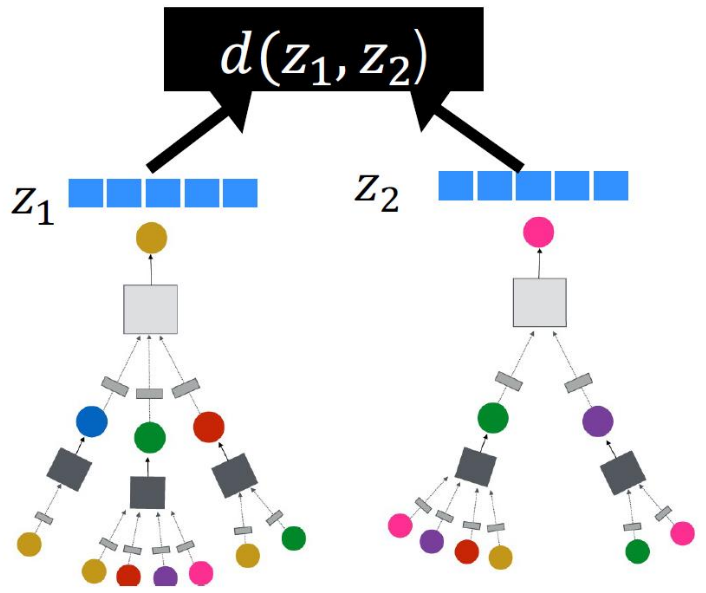
- In addition to the GNN model, the PinSAGE paper introduces several methods to scale the GNN to a billion-scale recommender system (e.g., Pinterest).
  - Shared negative samples across users in a mini-batch
  - Hard negative samples
  - Curriculum learning
  - Mini-batch training of GNNs on a large-graph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

# PinSAGE Model



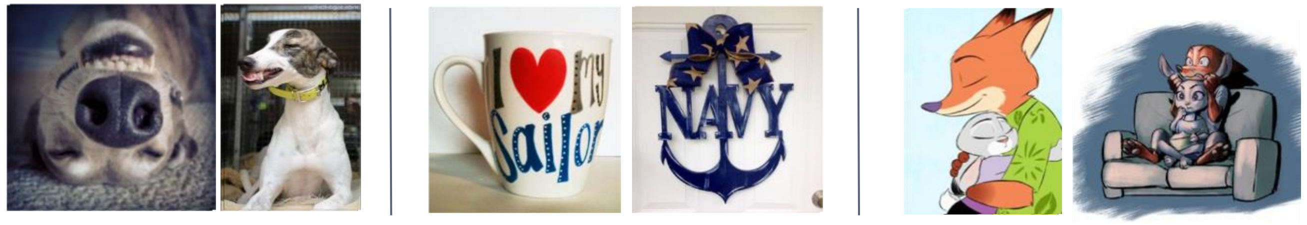
- **Task: Recommend related pins to users.**
- Learn node embeddings $z_i$ such that

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



# Training Data

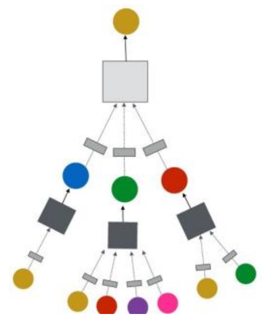
- 1+B repin pairs:
  - From Related Pins surface
  - Capture semantic relatedness
  - Goal: Embed such pairs to be “neighbors”
- Example positive training pairs (Q,X):

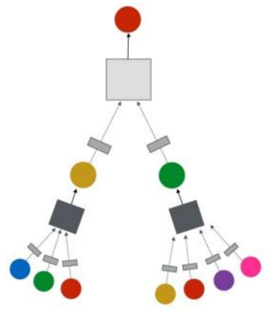


## Shared Negative Sample

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

- Similar to KG embedding, we can build a loss function for the positive pair and sampled negative pairs
- Using more negative samples $\boldsymbol{V}_{\text{neg}} = \{v_{\text{neg}}\}$ for one positive sample $v_{\text{pos}}$ per user $u^* \in \boldsymbol{U}_{\text{mini}}$ improves the recommendation performance, but is also expensive.
  - We need to generate $|\boldsymbol{U}_{\text{mini}}| \cdot |\boldsymbol{V}_{\text{neg}}|$ embeddings for negative nodes.
  - We need to apply $|\boldsymbol{U}_{\text{mini}}| \cdot |\boldsymbol{V}_{\text{neg}}|$ GNN computational graphs (see right), which is expensive.





## Shared Negative Sample

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

## 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.

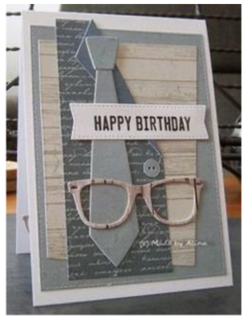
## PinSAGE: Curriculum Learning

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

Source pin

Positive

Easy negative



Hard negative

## Hard Negatives

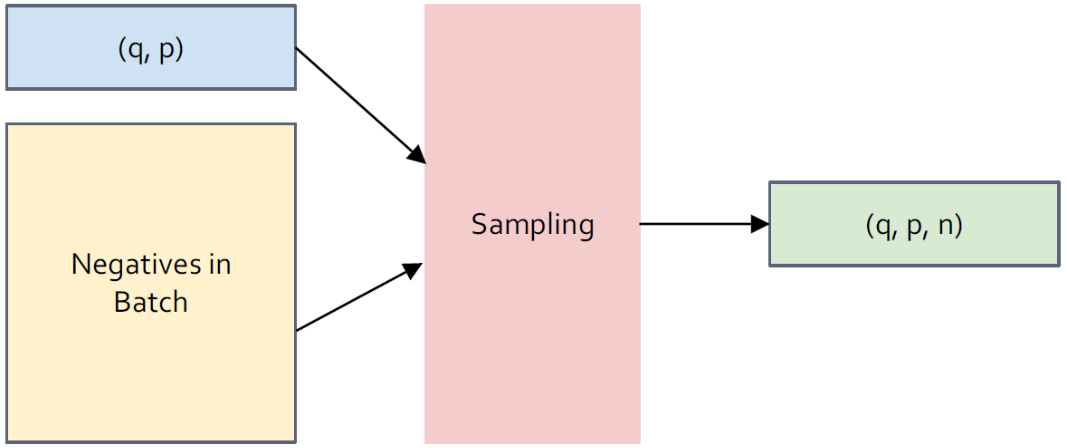
- **Challenge**: Industrial recsys needs to make **extremely fine-grained predictions**.
  - #Total items: Up to billions.
  - #Items to recommend for each user: 10 to 100.
- **Issue**: The shared negative items are  randomly sampled from all items
  - Most of them are "**easy negatives**", i.e., a model  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!

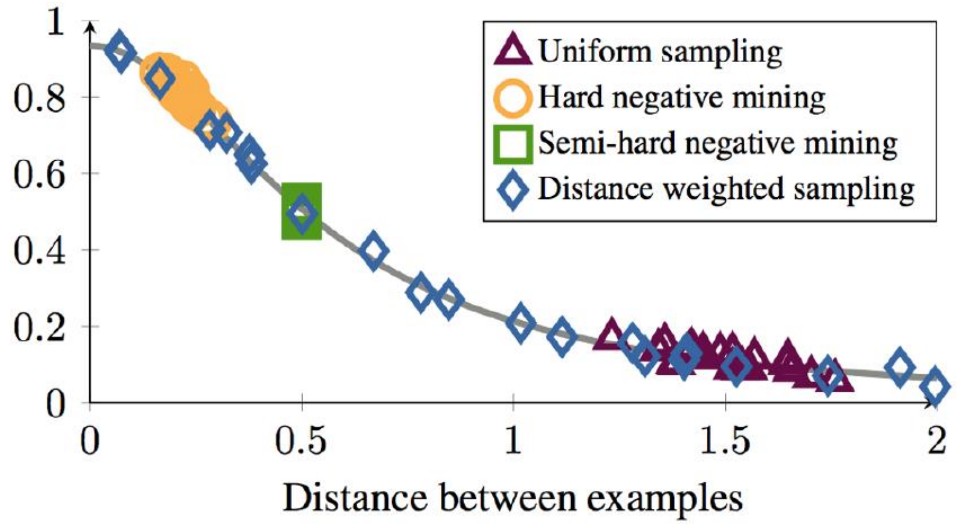
## Hard Negatives

- **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 $u \in \boldsymbol{U}$ 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., $2000^{\text{th}}$ —$5000^{\text{th}}$ .
    - 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

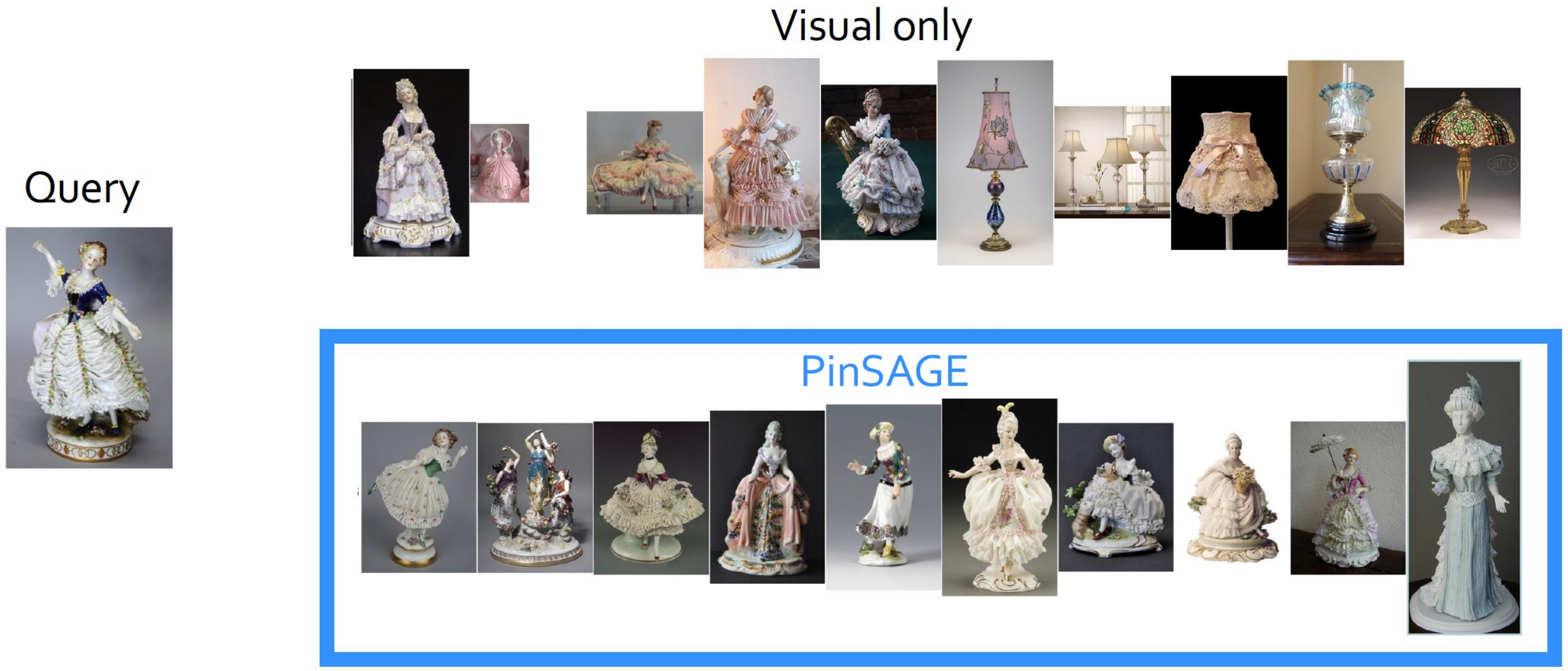
- (q, p) positive pairs are given but various methods to sample negatives to form (q, p, n)
- Distance Weighted Sampling (Wu et al., 2017)
    - Sample negatives so that query-negative distance distribution is approx U[0.5, 1.4]





(b) Sample distribution for different strategies.

# Fine-Grained Object Similarity



## 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.