# Stanford CS224W: Recommender Systems: Task and Evaluation

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

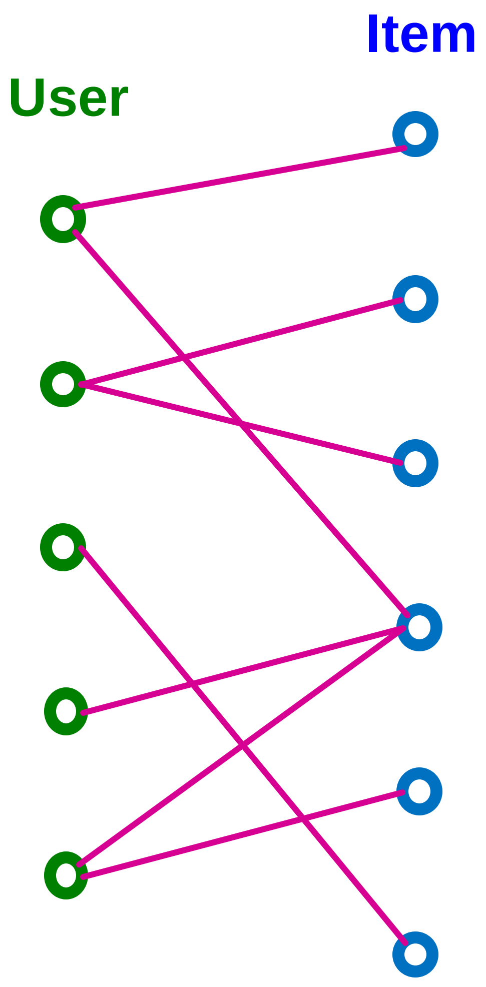
http://cs224w.stanford.edu

## 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.

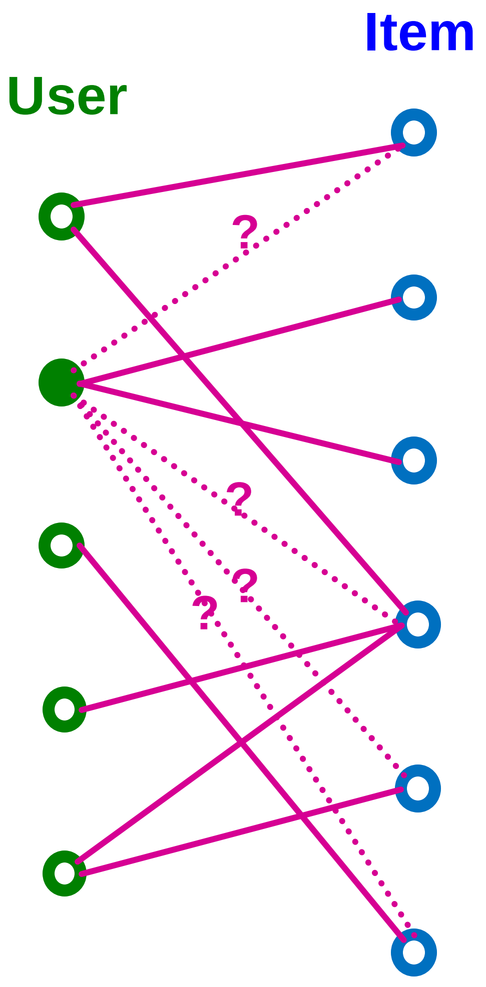
# Recommender System as a Graph

- 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

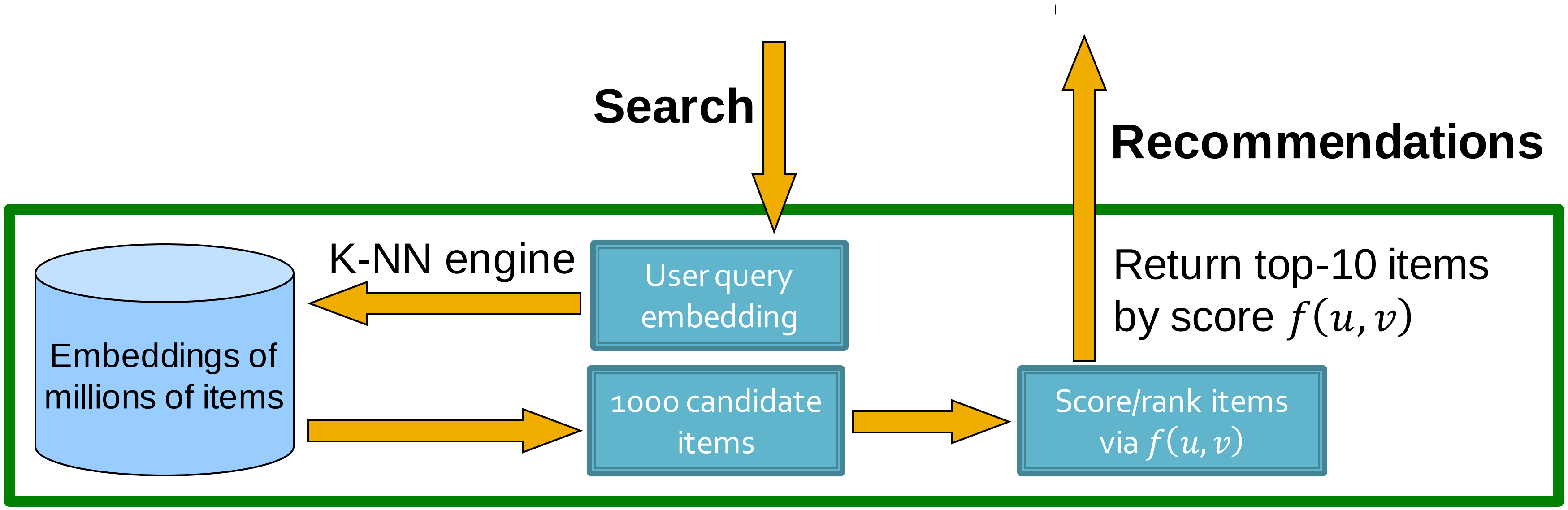
- **Given**
  - Past user-item interactions
- **Task**
  - Predict new items each user will interact in the future.
  - Can be cast as **link prediction** problem.
    - Predict new user-item interaction edges given the past edges.
  - For $u \in \boldsymbol{U}, v \in \boldsymbol{V}$, we need to get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



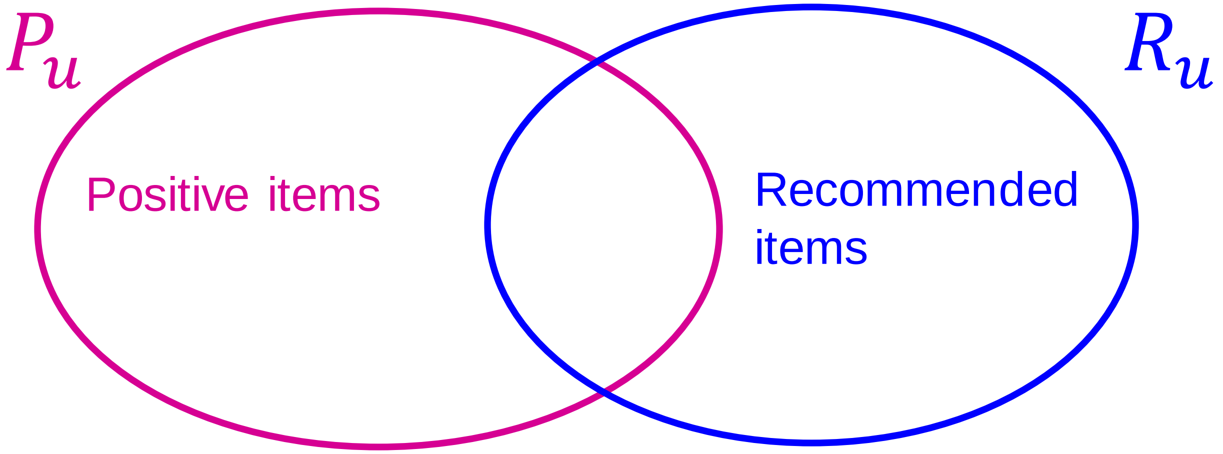
Search
Recommendations
K-NN engine
User query embedding
Embeddings of millions of items
1000 candidate items
Score/rank items via $f(u, v)$
Return top-10 items by score $f(u, v)$

## 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)

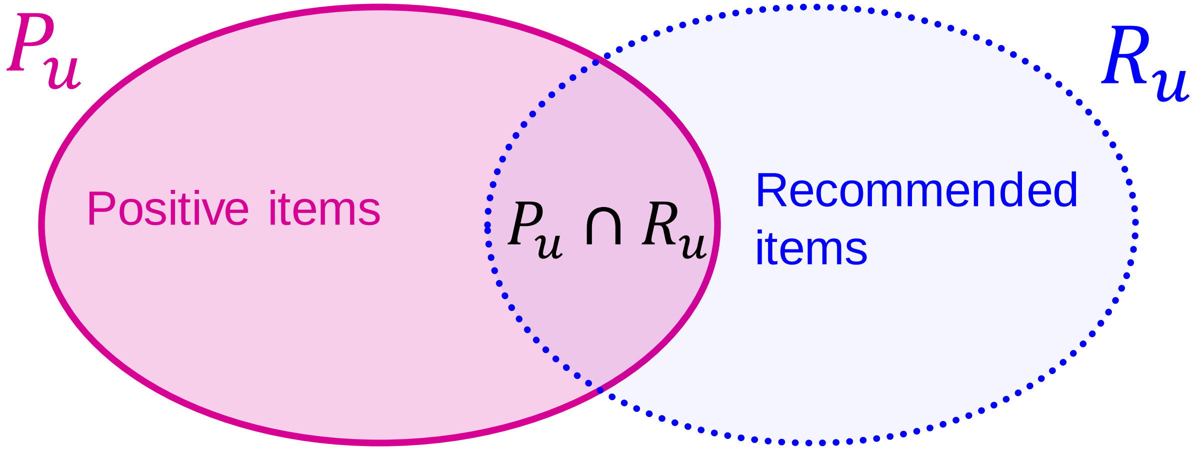
# Evaluation Metric: Recall@$K$ (1)

- **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.

# Stanford CS224W: Recommender Systems: Embedding-Based Models

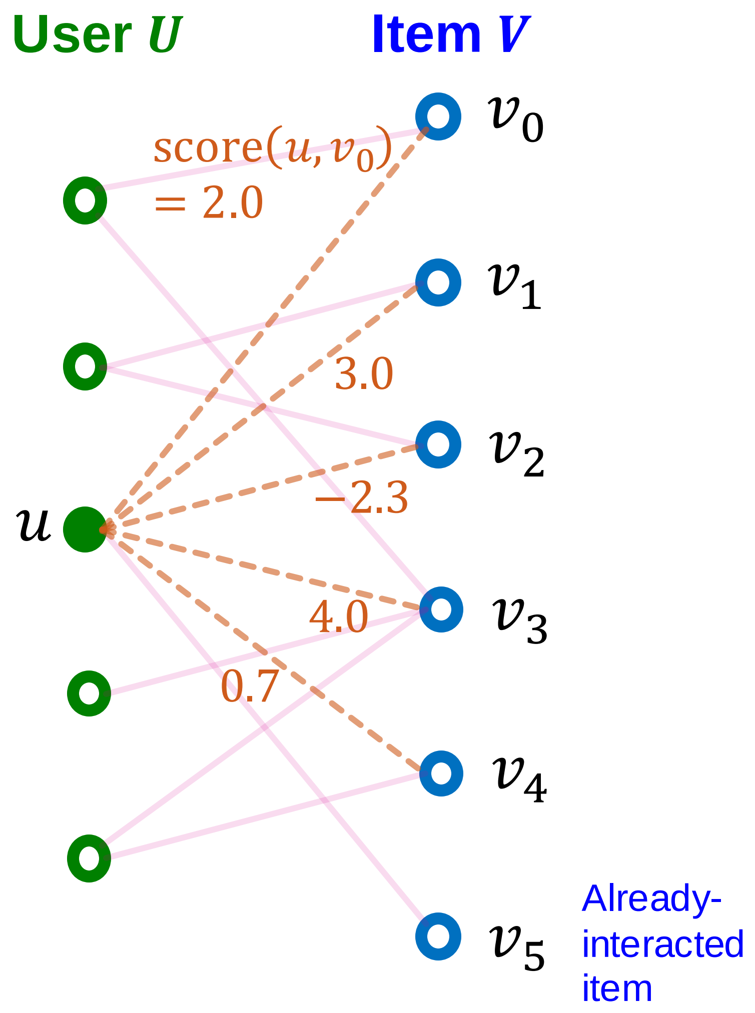
CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

## Notation

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

# Score Function

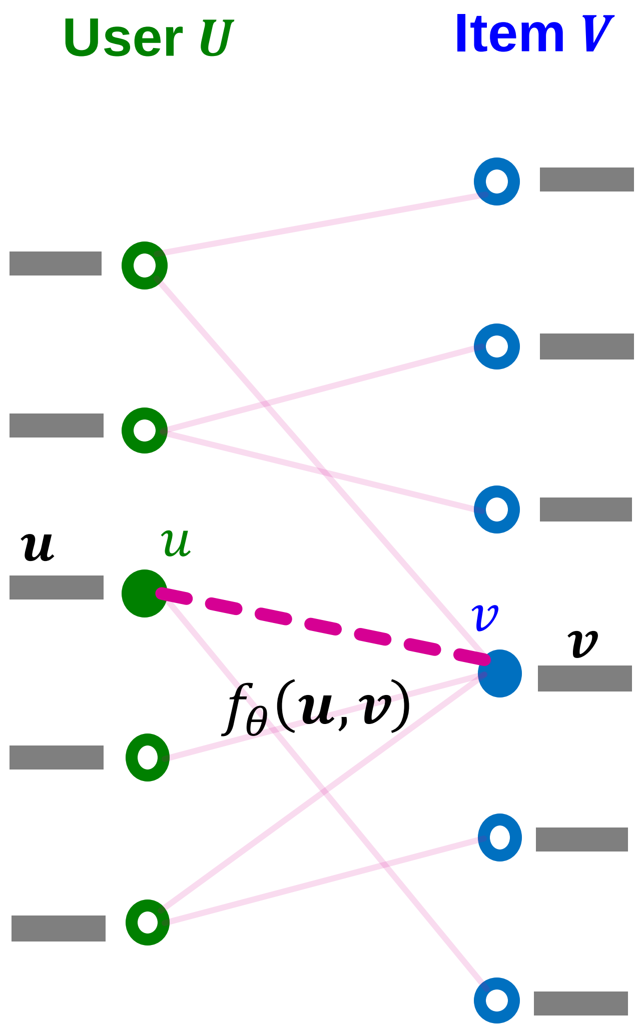
- To get the top-$K$ items, we need a score function for user-item interaction:
  - For $u \in \boldsymbol{U}, v \in \boldsymbol{V}$, we need to get a real-valued scalar **score**$(u, v)$.
  - ***K* items with the largest scores for a given user $u$** (excluding already-interacted items) are then recommended.



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 $f_\theta(\cdot,\cdot): \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ be a parametrized function.
  - Then, $\boxed{\text{score}(u, v) \equiv f_\theta(\boldsymbol{u}, \boldsymbol{v})}$



# Training Objective

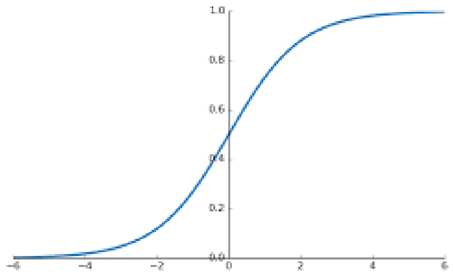
- Embedding-based models have three kinds of parameters:
  - An encoder to generate user embeddings $\{\boldsymbol{u}\}_{u \in U}$
  - An encoder to generate item embeddings $\{\boldsymbol{v}\}_{v \in V}$
  - Score function $f_\theta(\cdot,\cdot)$
- **Training objective**: Optimize the model parameters 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.

## 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**.

## Binary Loss (1)

- Define **positive**/**negative** **edges**
  - A set of **positive edges $\boldsymbol{E}$** (i.e., observed/training user-item interactions)
  - A set of **negative edges** $\boldsymbol{E}_{\mathbf{neg}} = \{(u, v) \mid (u, v) \notin E, u \in \boldsymbol{U}, v \in \boldsymbol{V}\}$
- Define **sigmoid function** $\sigma(x) \equiv \frac{1}{1+\exp(-x)}$
  - Maps real-valued scores into binary likelihood scores, i.e., in the range of [0,1].



## Binary Loss (2)

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

$$\underbrace{-\frac{1}{|\boldsymbol{E}|}\sum_{(u,v)\in\boldsymbol{E}}\log\left(\sigma(f_\theta(\boldsymbol{u},\boldsymbol{v}))\right)}\ -\ \underbrace{\frac{1}{|\boldsymbol{E}_{\mathbf{neg}}|}\sum_{(u,v)\in\boldsymbol{E}_{\mathbf{neg}}}\log\left(1-\sigma(f_\theta(\boldsymbol{u},\boldsymbol{v}))\right)}$$

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

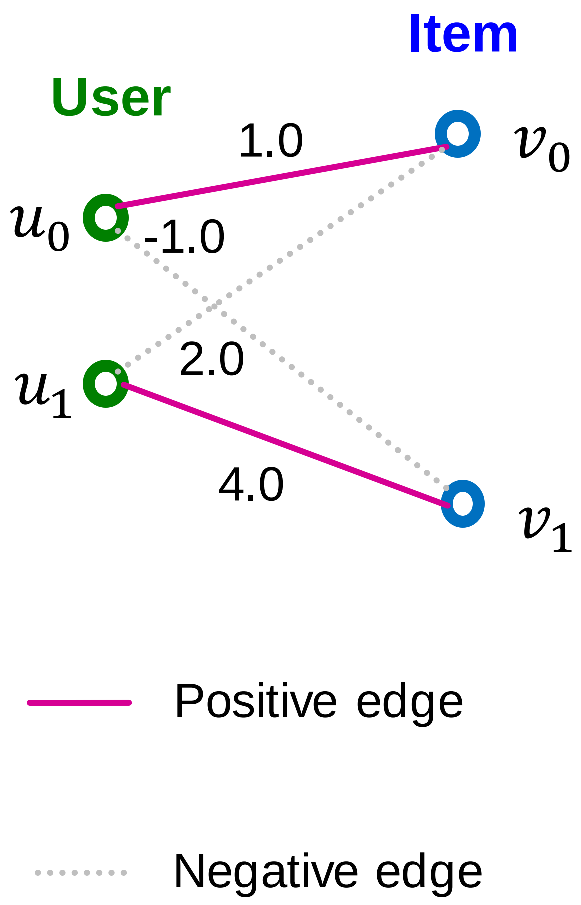
- Binary loss pushes the scores of **positive edges** higher than those of **negative edges**.
  - This aligns with the training recall metric since positive edges need to be recalled.

# Issue with Binary Loss (1)

- Issue: In the binary loss, the scores of ***ALL*** positive edges are pushed higher than those of ***ALL*** negative edges.
- This would unnecessarily penalize model predictions even if the training recall metric is perfect.
- **Why?** (example in the next slide)

# Issue with Binary Loss (2)

- **Let's consider the simplest case:**
  - Two users, two items
  - Metric: Recall@1.
  - A model assigns the score for every user-item pair (as shown in the right).
- Training **Recall@1 is 1.0** (perfect score), because $v_0$ (resp. $v_1$) is correctly recommended to $u_0$ (resp. $u_1$).
- However, **the binary loss would still penalize the model prediction** because the negative $(u_1, v_0)$ edge gets the higher score than the positive edge $(u_0, v_0)$.

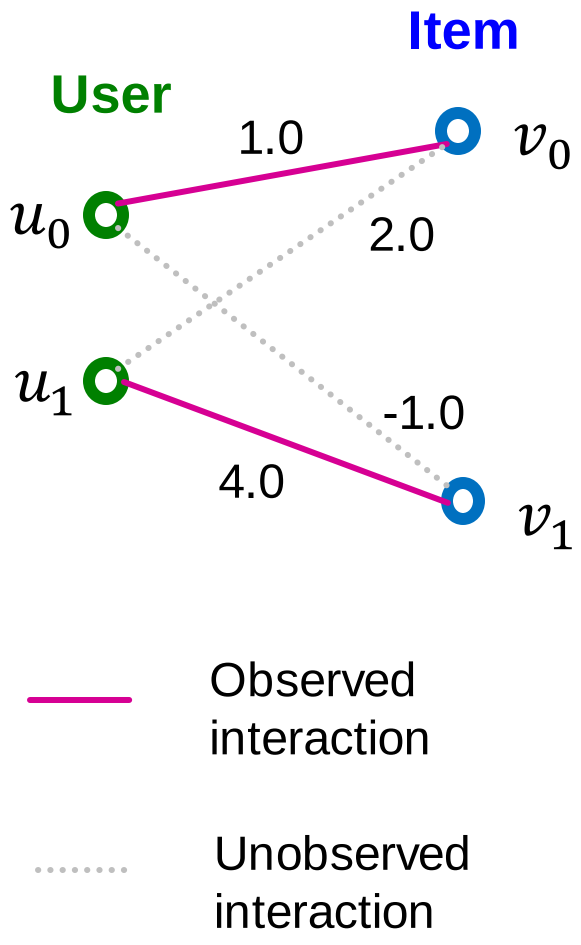


# Issue with Binary Loss (3)

- **Key insight**: The binary loss is **non-personalized** in the sense that the **positive/negative edges are considered *across ALL users at once*.**
- However, the recall metric is inherently **personalized (defined for each user)**.
  - The non-personalized binary loss is overly-stringent for the personalized recall metric.

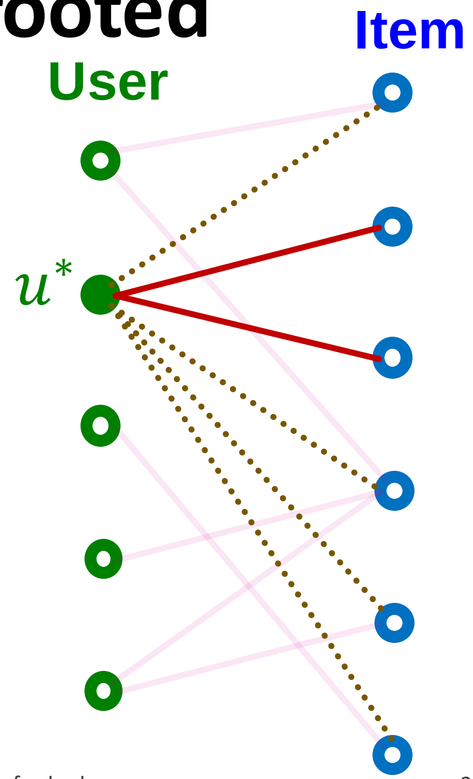
# Desirable Surrogate Loss

- **Lesson learned**: Surrogate loss function should be defined in a **personalized** manner.
  - **For each user**, we want the scores of positive items to be higher than those of the negative items
  - We do *not* care about the score ordering across users.
- **Bayesian Personalized Ranking (BPR)** loss achieves this!



# Loss Function: BPR Loss (1)

- **Bayesian Personalized Ranking (BPR) loss** is a personalized surrogate loss that aligns better with the recall@K metric.
- For each user $u^* \in \boldsymbol{U}$, define the **rooted positive/negative edges** as
  - Positive edges rooted at $u^*$
    - $\boldsymbol{E}(u^*) \equiv \{(u^*, v) \mid (u^*, v) \in \boldsymbol{E}\}$
  - Negative edges rooted at $u^*$
    - $\boldsymbol{E}_{\mathbf{neg}}(u^*) \equiv \{(u^*, v) \mid (u^*, v) \in \boldsymbol{E}_{\mathbf{neg}}\}$



Note: The term "Bayesian" is not essential to the loss definition. The original paper [Rendle et al. 2009] considers the Bayesian prior over parameters (essentially acts as a parameter regularization), which we omit here.

## Loss Function: BPR Loss (2)

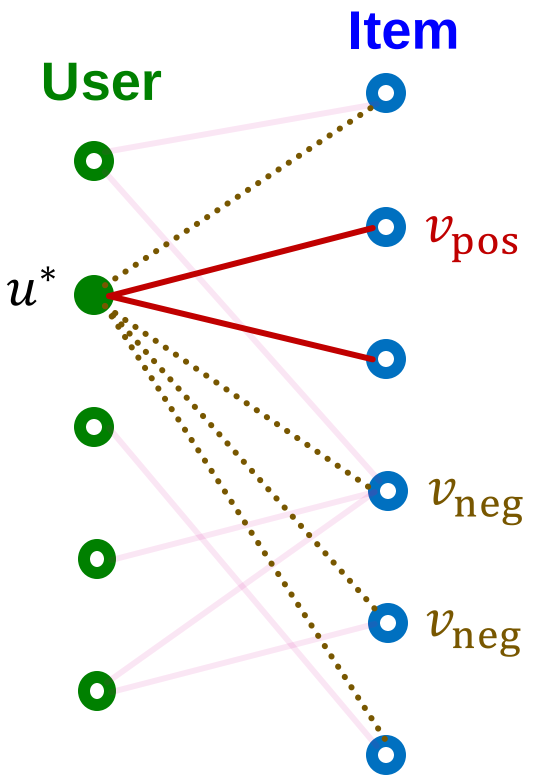
- **Training objective**: **For each user** $u^*$, we want the scores of rooted positive edges $\boldsymbol{E}(u^*)$ to be higher than those of rooted negative edges $\boldsymbol{E}_{\mathbf{neg}}(u^*)$.
  - **Aligns with the personalized nature of the recall metric.**
- **BPR Loss for user $\boldsymbol{u}^*$:**

$$\mathrm{Loss}(u^*) = \frac{1}{|\boldsymbol{E}(u^*)| \cdot |\boldsymbol{E}_{\mathbf{neg}}(u^*)|} \underbrace{\sum_{(u^*, v_{\mathrm{pos}}) \in \boldsymbol{E}(u^*)} \sum_{(u^*, v_{\mathrm{neg}}) \in \boldsymbol{E}_{\mathbf{neg}}(u^*)} -\log\left(\sigma\left(\overbrace{f_\theta(\boldsymbol{u}^*, \boldsymbol{v}_{\mathrm{pos}}) - f_\theta(\boldsymbol{u}^*, \boldsymbol{v}_{\mathrm{neg}})}^{\text{Encouraged to be positive for each user =positive edge score is higher than negative edge score}}\right)\right)}_{\text{Can be approximated using a mini-batch}}$$

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

# Loss Function: BPR Loss (3)

- **Mini-batch training for the BPR loss:**
  - In each mini-batch, we sample a subset of users $\boldsymbol{U}_{\text{mini}} \subset \boldsymbol{U}$.
    - For each user $u^* \in \boldsymbol{U}_{\text{mini}}$, we sample one positive item $v_{\text{pos}}$ and a set of sampled negative items $V_{\text{neg}} = \{v_{\text{neg}}\}$.
  - The mini-batch loss is computed as



$$\boxed{\frac{1}{|\boldsymbol{U}_{\text{mini}}|} \sum_{u^* \in \boldsymbol{U}_{\text{mini}}}} \frac{1}{|\boldsymbol{V}_{\text{neg}}|} \sum_{v_{\text{neg}} \in \boldsymbol{V}_{\text{neg}}} -\log\Big(\sigma\big(f_\theta(u^*, \boldsymbol{v}_{\text{pos}}) - f_\theta(u^*, \boldsymbol{v}_{\text{neg}})\big)\Big)$$

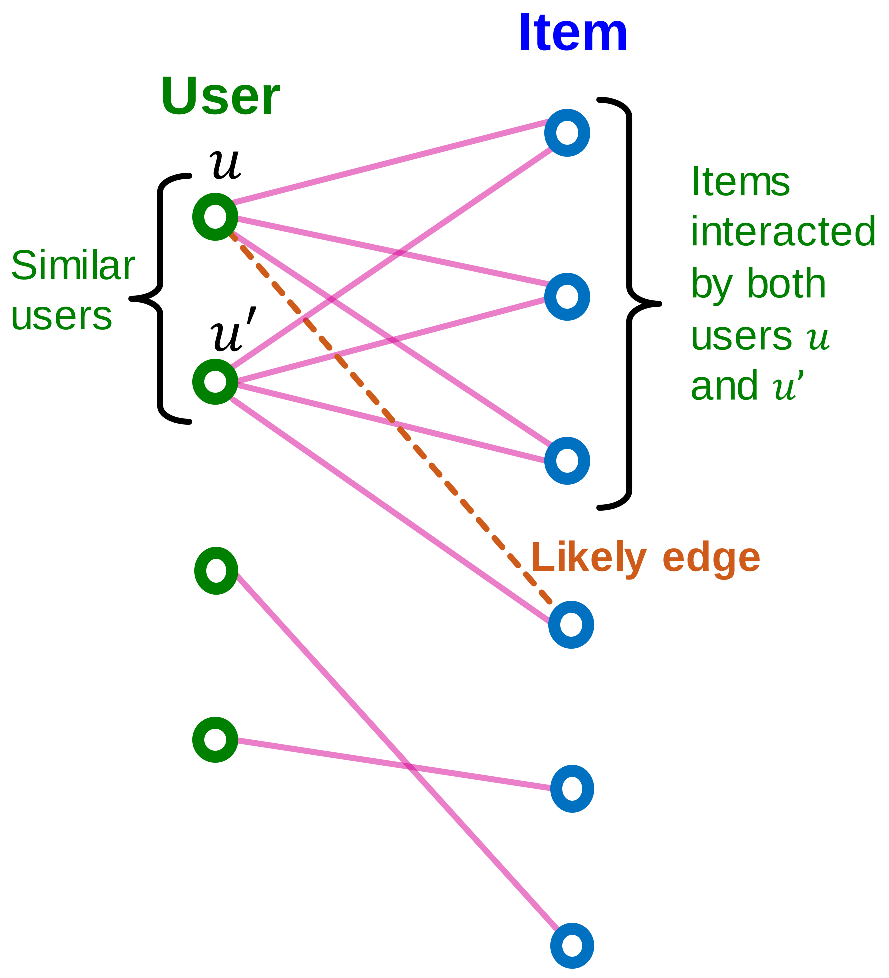
**Average over users in the mini-batch**

## 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

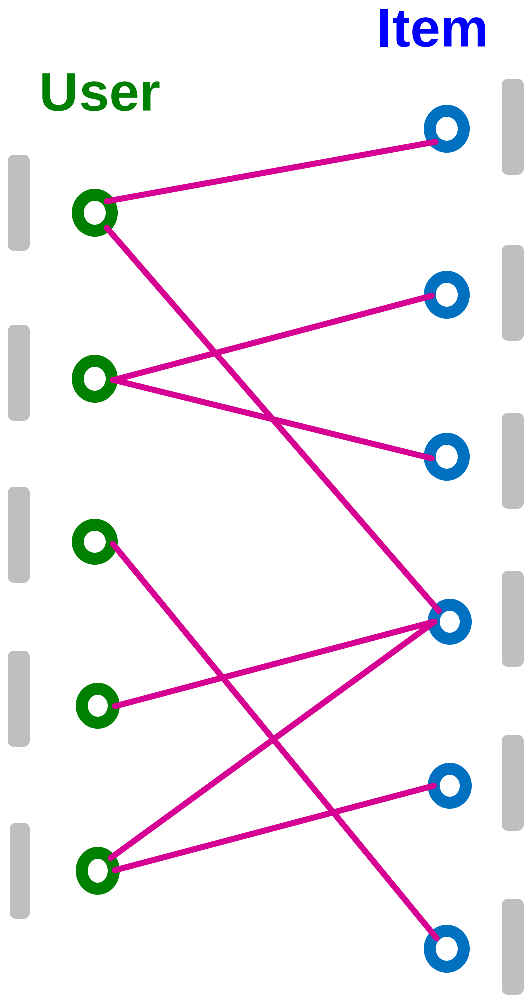
- **In this lecture, we teach two representative GNN approaches for recommender systems.**
- **(1) Neural Graph Collab. Filtering (NGCF)** [Wang et al. 2019]
- **(2) LightGCN** [He et al. 2020]
  - Improve the conventional collaborative filtering models (i.e., shallow encoders) by explicitly modeling 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.

# Stanford CS224W: Neural Graph Collaborative Filtering

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

# 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 $u \in \boldsymbol{U}$ 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

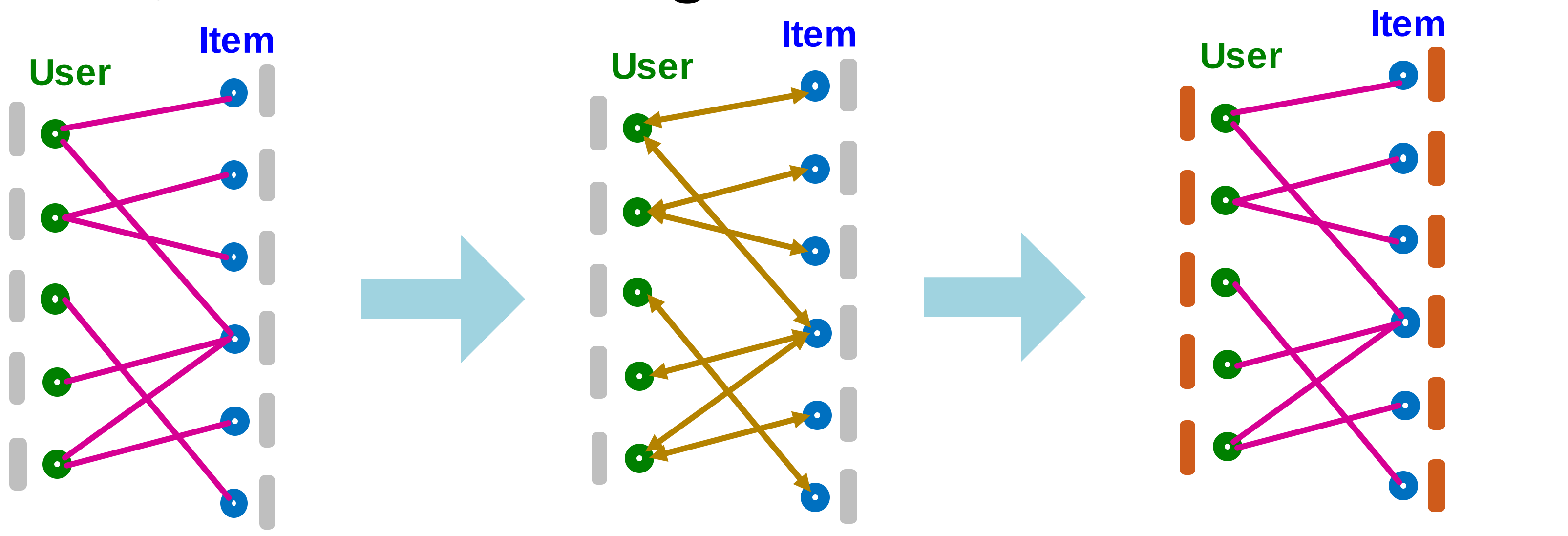
- The model itself does ***not explicitly*** capture graph structure
  - The graph structure is ***only implicitly*** captured in the training objective.
- Only the **first-order graph structure** (i.e., edges) is captured in the training objective.
  - **High-order graph structure** (e.g., $K$-hop paths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

## 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.



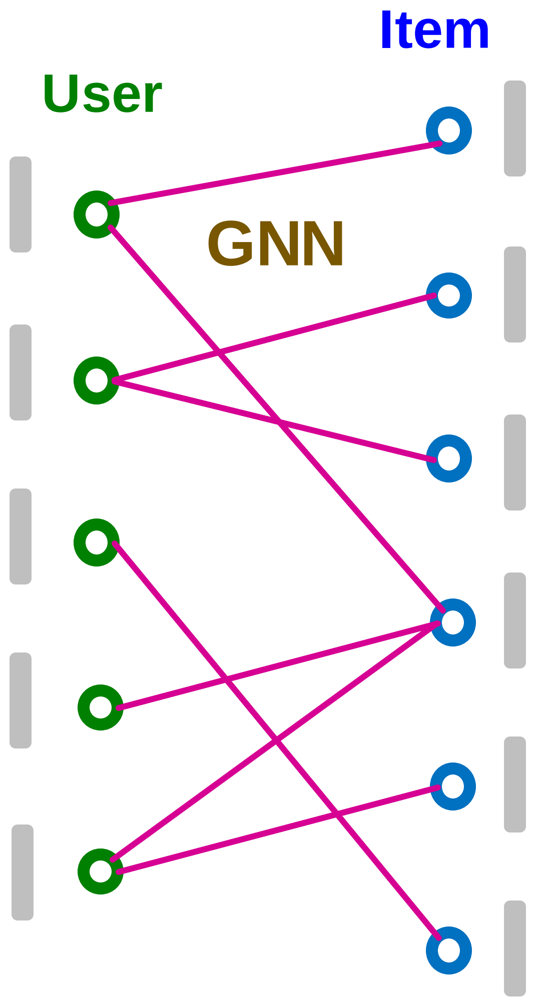
Initial shallow embeddings (not graph-aware)

Use a GNN to propagate embeddings

NGCF's graph-aware embeddings

# NGCF Framework

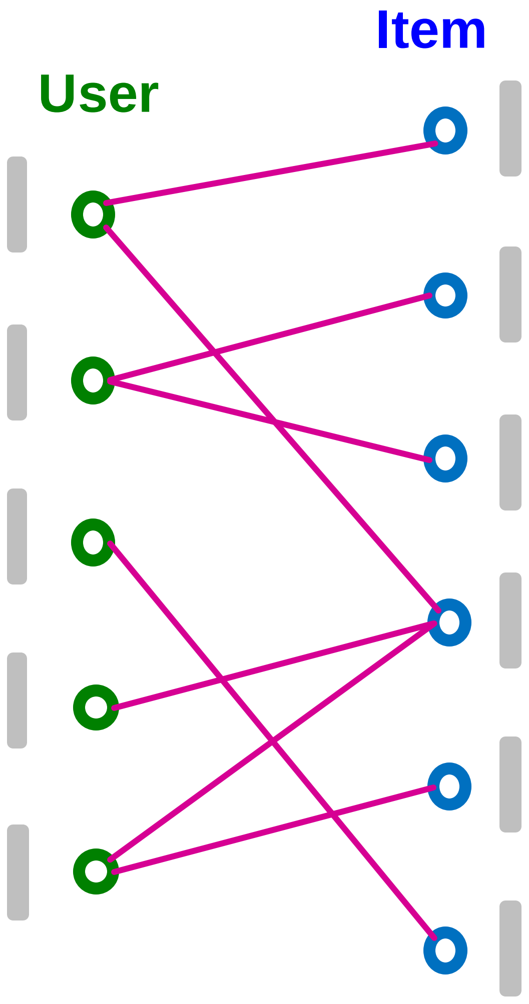
- **Given**: User-item bipartite graph.
- **NGCF framework:**
  - Prepare shallow learnable embedding for each node.
  - Use multi-layer GNNs to propagate embeddings along the bipartite graph.
    - High-order graph structure is captured.
  - Final embeddings are *explicitly* graph-aware!
- **Two kinds of learnable params 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 $\boldsymbol{h}_u^{(0)}$ 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

# Neighbor Aggregation

- Iteratively update node embeddings using neighboring embeddings.

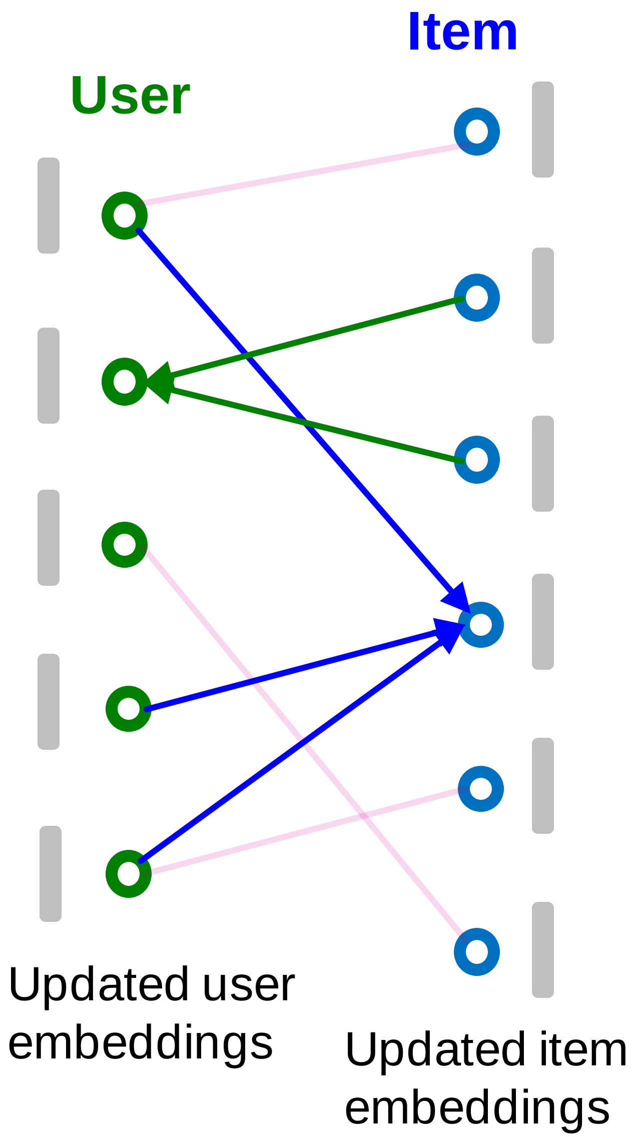
$$h_v^{(k+1)} = \text{COMBINE}\left(h_v^{(k)}, \text{AGGR}\left(\left\{h_u^{(k)}\right\}_{u\in N(v)}\right)\right)$$

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

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

Different architecture choices are possible for AGGR and COMBINE.

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



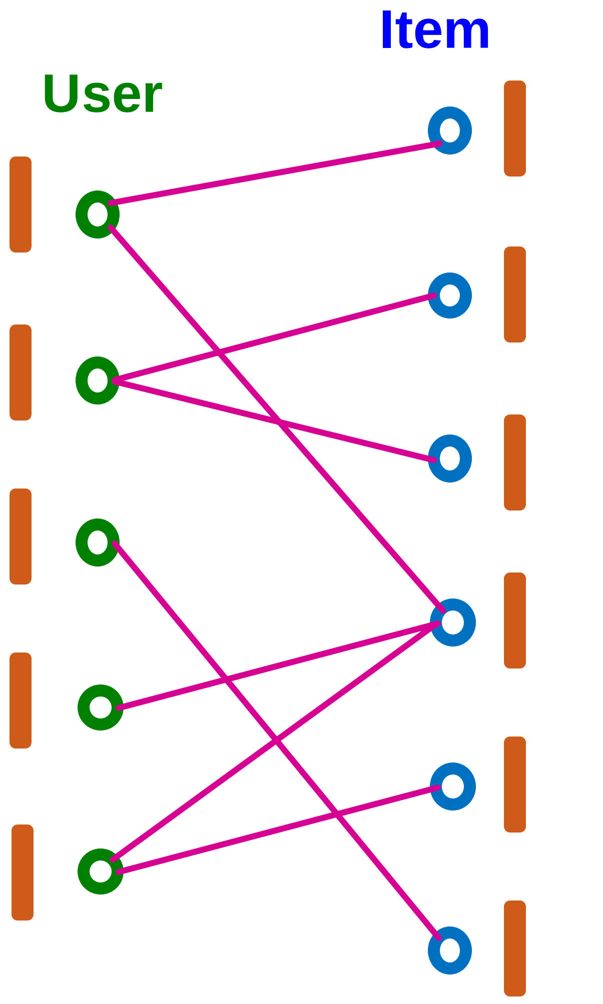
# Final Embeddings and Score Function

- After $K$ rounds of neighbor aggregation, we get the **final user/item embeddings** $\boldsymbol{h}_u^{(K)}$ 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

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



Final user/item embeddings (graph-aware)

## 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.**

# Stanford CS224W: LightGCN

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

# 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]

# Adjacency and Embedding Matrices

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

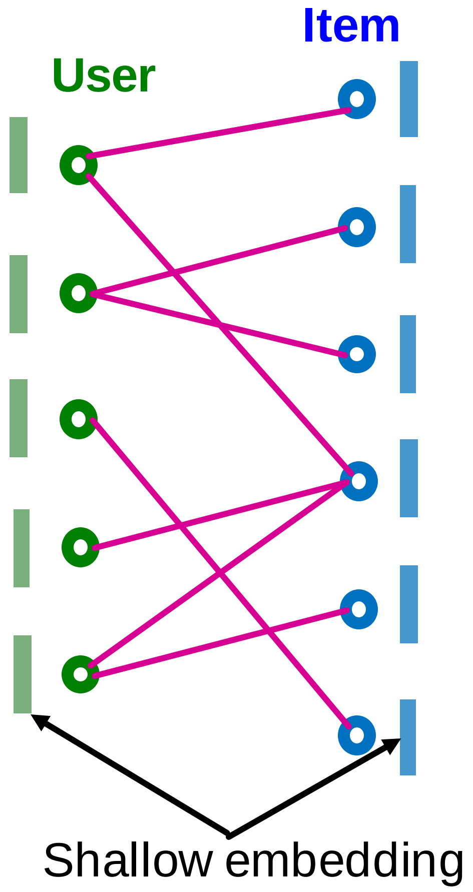
Adjacency matrix $A$

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

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

Embedding matrix $\boldsymbol{E}$

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



# Matrix Formulation of GCN

- **Recall**: The diffusion matrix of C&S.
- 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.

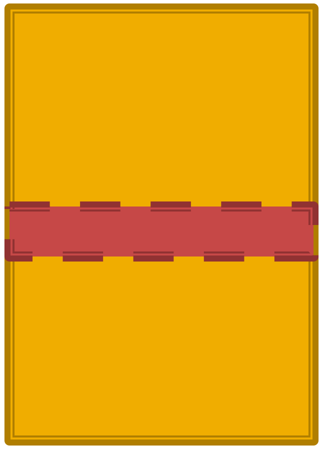
- Let $\boldsymbol{E}^{(k)}$ be the embedding matrix at $k$-th layer.
- Each layer of GCN's aggregation can be written in a matrix form:

$$\boldsymbol{E}^{(k+1)} = \text{ReLU}\left(\widetilde{\boldsymbol{A}}\boldsymbol{E}^{(k)}\boldsymbol{W}^{(k)}\right)$$

**Neighbor aggregation**

**Learnable linear transformation**

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



**Each row stores node embedding**

# Simplifying GCN (1)

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

$$\boldsymbol{E}^{(k+1)} = \widetilde{\boldsymbol{A}}\boldsymbol{E}^{(k)}\boldsymbol{W}^{(k)}$$

**Original idea from SGC [Wu et al. 2019]**

- 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 $\boldsymbol{E}^{(0)}$

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

## Simplifying GCN (2)

- Removing ReLU significantly simplifies GCN!

$$\boldsymbol{E}^{(K)} = \boxed{\widetilde{\boldsymbol{A}}^{K}\ \boldsymbol{E}}\ \boldsymbol{W}$$

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

**Diffusing node embeddings along the graph**
(similar to C&S that diffuses soft labels 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 $\widetilde{\boldsymbol{A}}^{K}\ \boldsymbol{E}$.

## Multi-Scale Diffusion

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

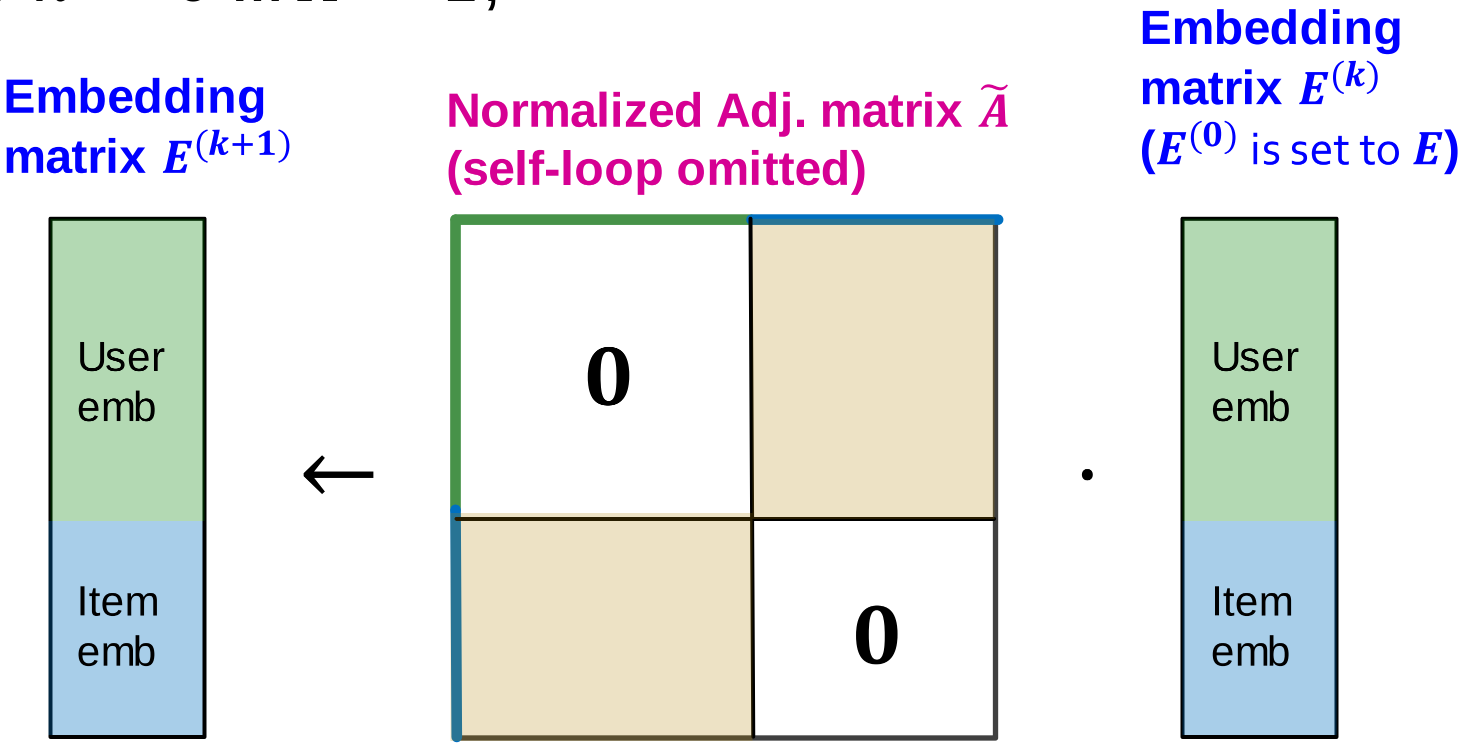
$$\alpha_0 E^{(0)} + \alpha_1 E^{(1)} + \alpha_2 E^{(2)} + \cdots + \alpha_K E^{(K)}$$

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

## LightGCN: Model Overview (2)

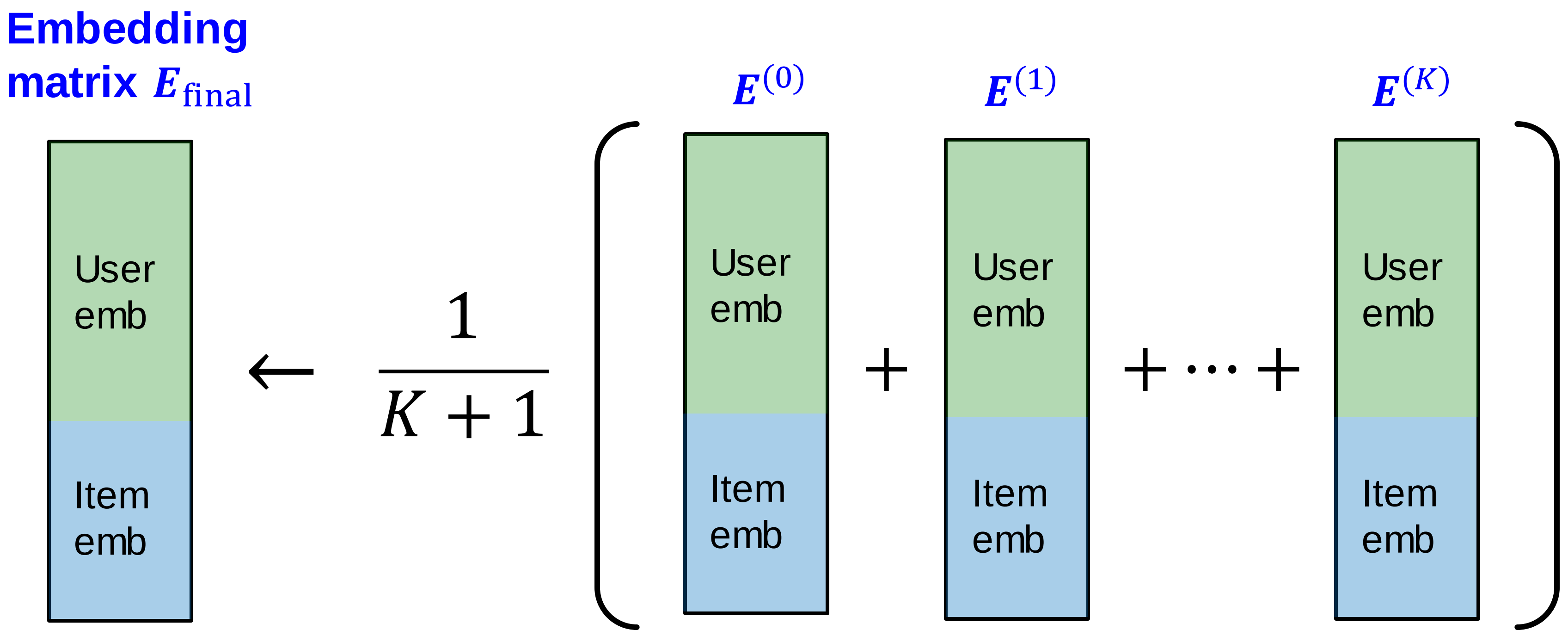
Iteratively diffuse embedding matrix $\boldsymbol{E}$ using $\widetilde{\boldsymbol{A}}$

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


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

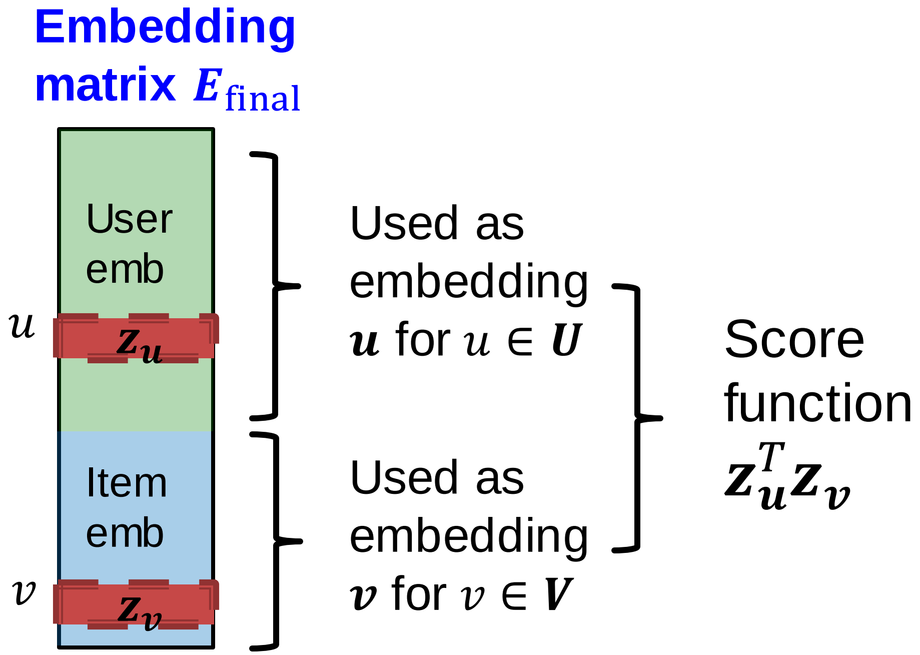
# LightGCN: Model Overview (3)

- Average the embedding matrices at different scales.



# LightGCN: Model Overview (4)

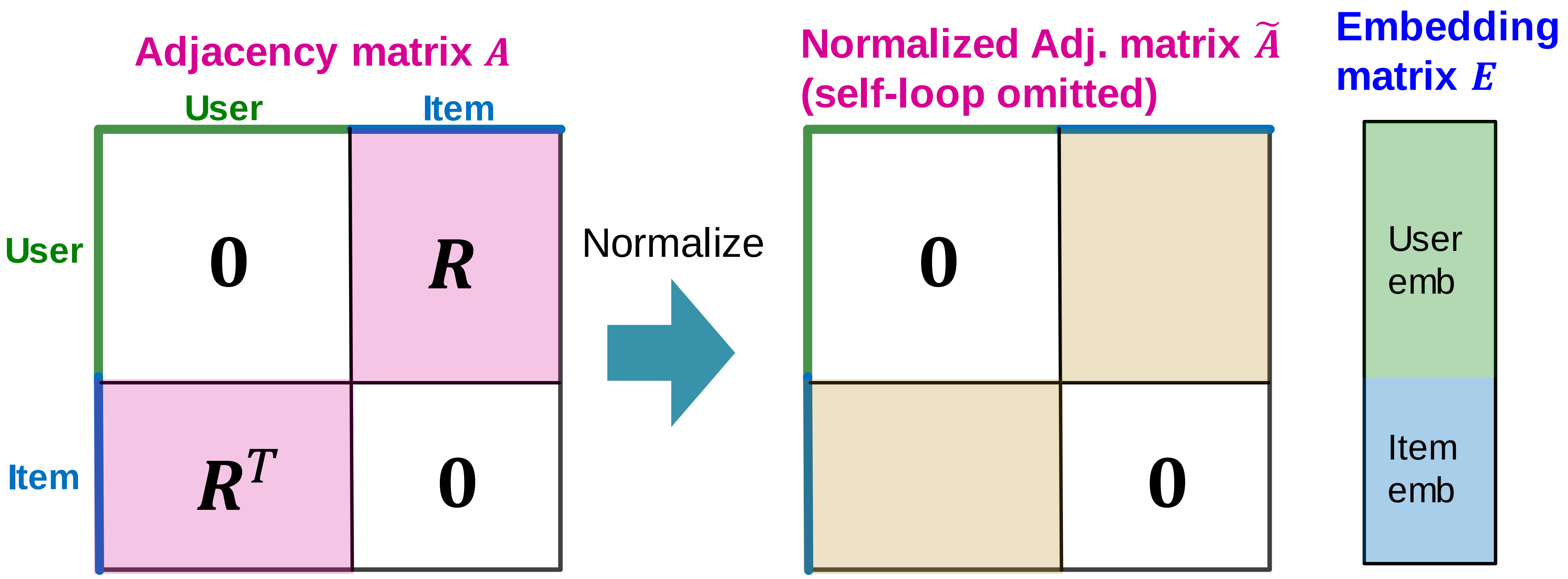
- **Score function:**
  - Use user/item vectors from $\boldsymbol{E}_{\text{final}}$ to score user-item interaction



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

# LightGCN: Model Overview (1)

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



## 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).

# LightGCN and GCN/C&S

- The embedding propagation of LightGCN is closely related to GCN/C&S.
- **Recall**: GCN/C&S (neighbor aggregation part)

$$\boldsymbol{h}_v^{(k+1)} = \sum_{u \in N(v)} \frac{1}{\sqrt{d_u}\boxed{\sqrt{d_v}}} \cdot \boldsymbol{h}_u^{(k)}$$

**Node degree**

  - Self-loop is added in the neighborhood definition.
- LightGCN uses the same equation except that
  - Self-loop is *not* added in the neighborhood definition.
  - Final embedding takes the average of embeddings from all the layers: $\boldsymbol{h}_v = \frac{1}{K+1}\sum_{k=0}^{K} \boldsymbol{h}_v^{(k)}$.

## LightGCN and MF: Comparison

- Both LightGCN and shallow encoders **learn a unique embedding for each user/item.**
- The difference is that LightGCN uses the ***diffused*** user/item embeddings for scoring.
- LightGCN performs better than shallow encoders 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.

## Stanford CS224W: PinSAGE

CS224W: Machine Learning with Graphs
Jure Leskovec, Stanford University
http://cs224w.stanford.edu

## PinSAGE: Pin Embedding

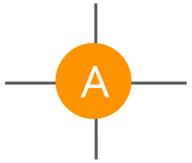
- Unifies visual, textual, and graph information.
- The largest industry deployment of a Graph Convolutional Networks.
- Huge Adoption across Pinterest
- Works for fresh content and is available in a few seconds after pin creation

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

# Application: Pinterest

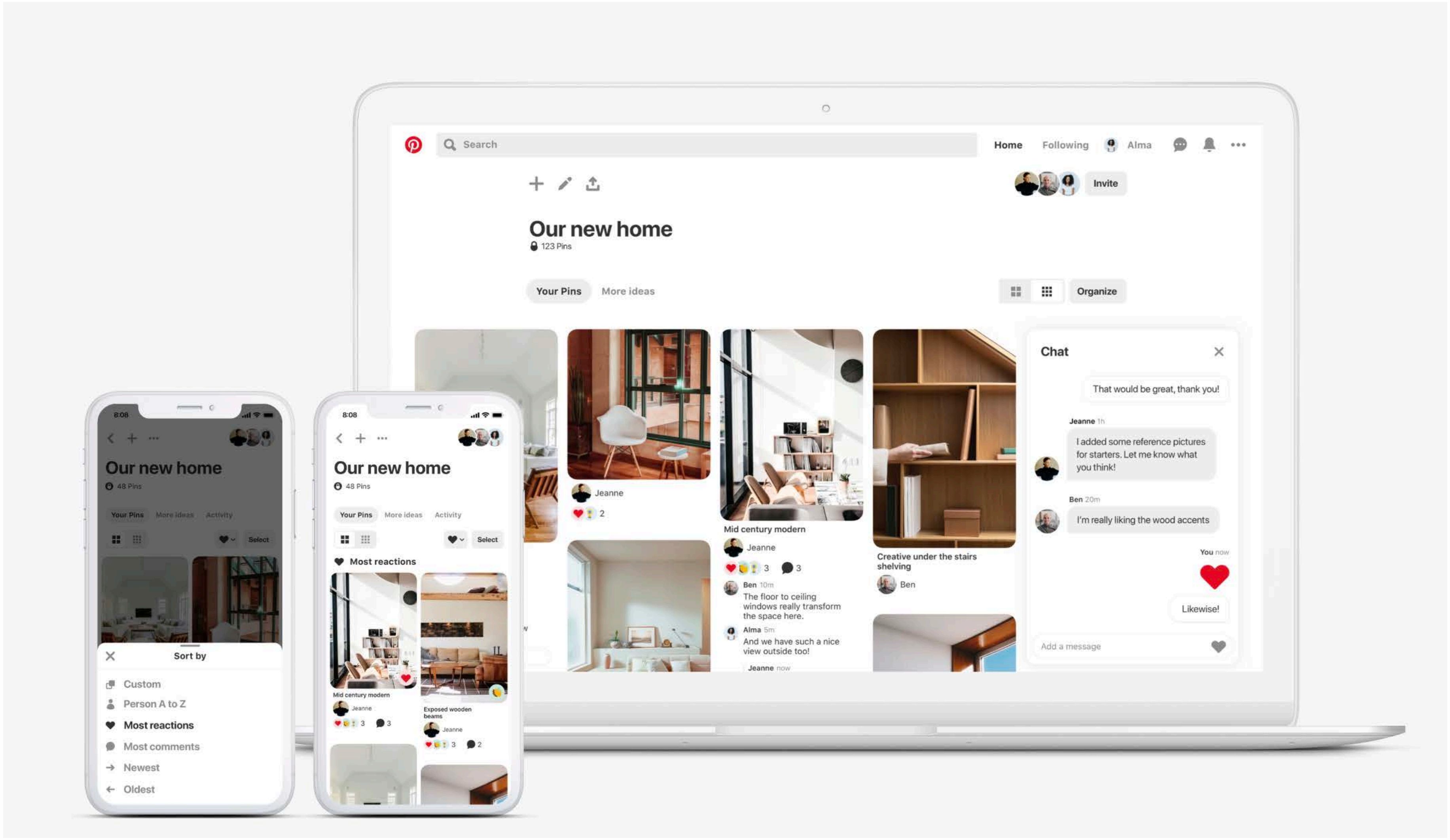
**PinSage graph convolutional network:**

- **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



Our new home
123 Pins

Mid century modern
Jeanne

Creative under the stairs shelving
Ben

Exposed wooden beams

Ben 10m
The floor to ceiling windows really transform the space here.

Alma 5m
And we have such a nice view outside too!

Chat

That would be great, thank you!

Jeanne 1h
I added some reference pictures for starters. Let me know what you think!

Ben 20m
I'm really liking the wood accents

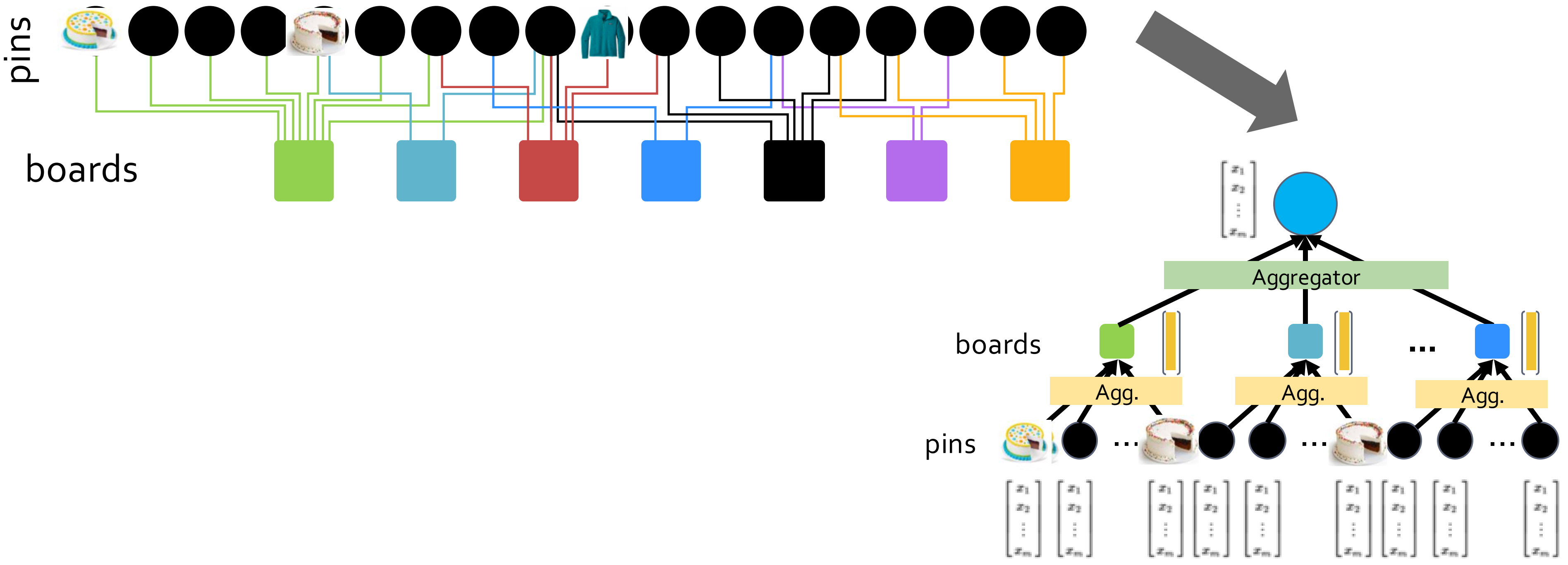
You now

Likewise!

*source: Pinterest newsroom*

# PinSAGE: Graph Neural Network

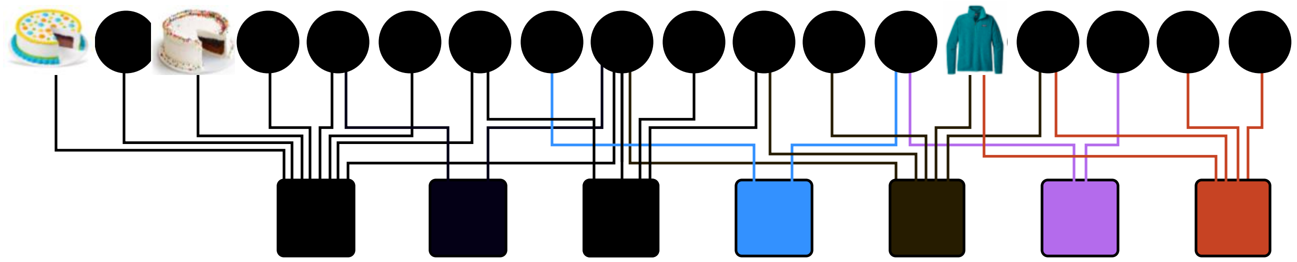
- Graph has tens of billions of nodes and edges
- Further resolves embeddings across the Pinterest graph



# 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)

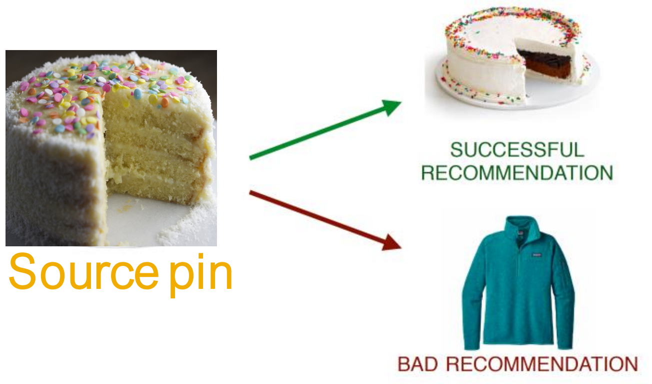
# PinSAGE Model

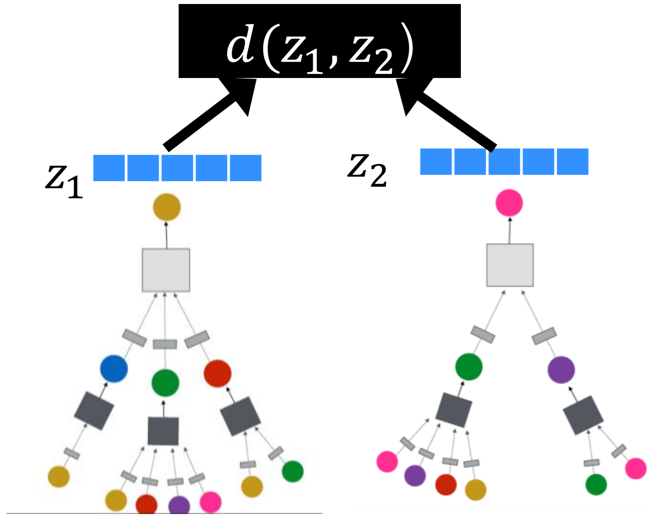


## Task: Recommend related pins to users

Learn node embeddings $z_i$ such that

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





## Training Data

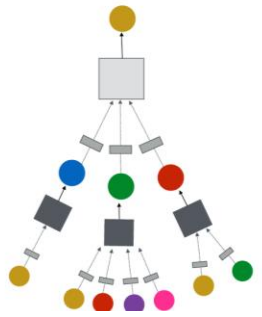
1+B repin pairs:

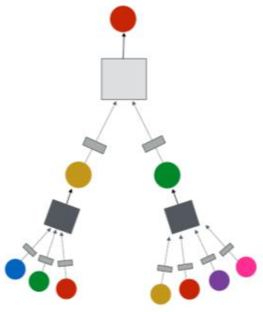
- From Related Pins surface
- Capture semantic relatedness
- Goal: Embed such pairs to be "neighbors"

Example positive training pairs (Q,X):

# Shared Negative Samples (1)

- **Recall**: In BPR loss, for each user $u^* \in \boldsymbol{U}_{\text{mini}}$, we sample one positive item $v_{\text{pos}}$ and a set of sampled negative items $\boldsymbol{V}_{\text{neg}} = \{v_{\text{neg}}\}$.
- Using more negative samples per user 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 Samples (2)

- **Key idea**: We can share the same set of negative samples $\boldsymbol{V}_{\text{neg}} = \{v_{\text{neg}}\}$ ***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** $|\boldsymbol{U}_{\text{mini}}|$!
  - 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.

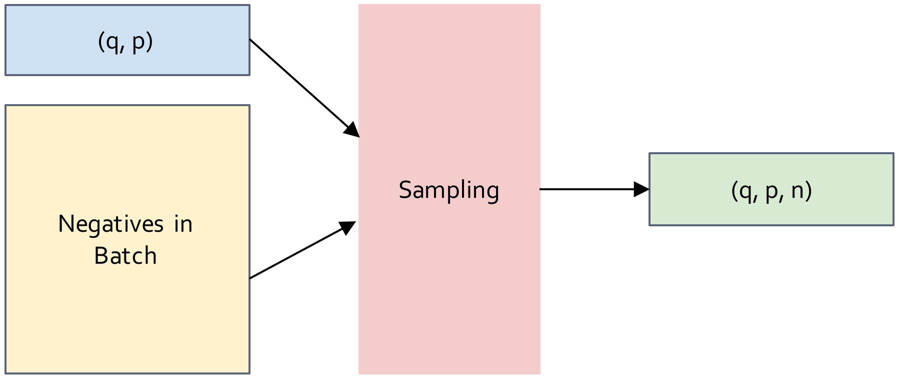
## Hard Negatives (2)

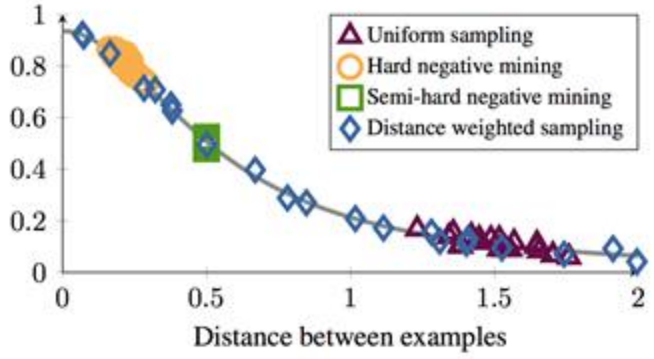
- **For each user node**, the **hard negatives** are item nodes that are close (but not connected) to the user node in the graph.
- Hard negatives for user $u \in \boldsymbol{U}$ are obtained as follows:
  - Compute personalized page rank (PPR) for user $u$.
  - Sort items in the descending order of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

# 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.