Stanford CS224W: GNNs for Recommender Systems

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

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.

Matrix Factorization

	Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
	✓		✓	✓	
		✓			✓
	✓	✓	✓		
				✓	✓

≈

Item embedding

.9	-1	1	1	-.9
-.2	-.8	-1	.9	1

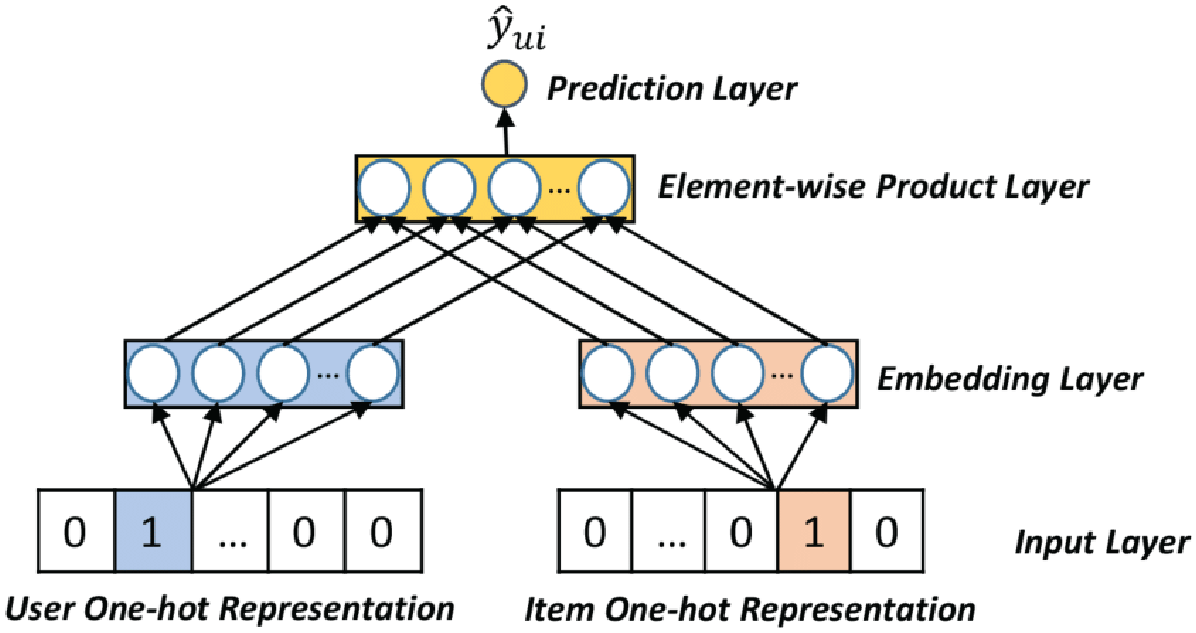
User embedding

1	.1
-1	0
.2	-1
.1	1

.88	-1.08	0.9	1.09	-0.8
-0.9	1.0	-1.0	-1.0	0.9
0.38	0.6	1.2	-0.7	-1.18
-0.11	-0.9	-0.9	1.0	0.91

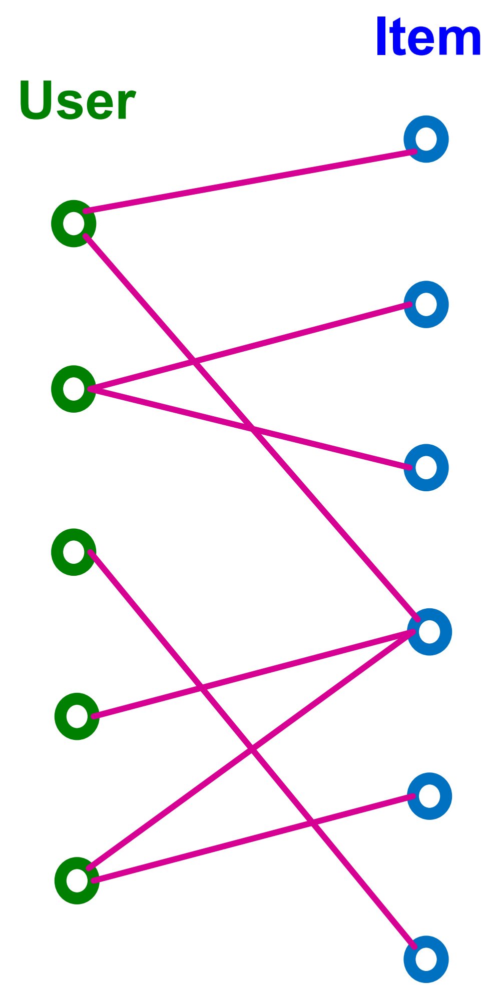
The embeddings are learned such that their dot product is a good approximation of the user-item matrix

Matrix-factorization as a shallow neural network model



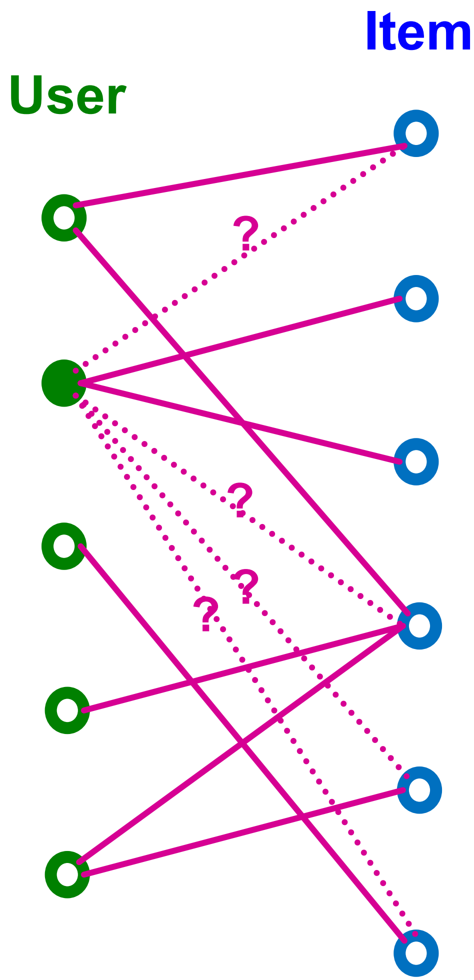
Recommender System as a Graph

- 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.



Stanford CS224W: Recommender Systems: Embedding-Based Models

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

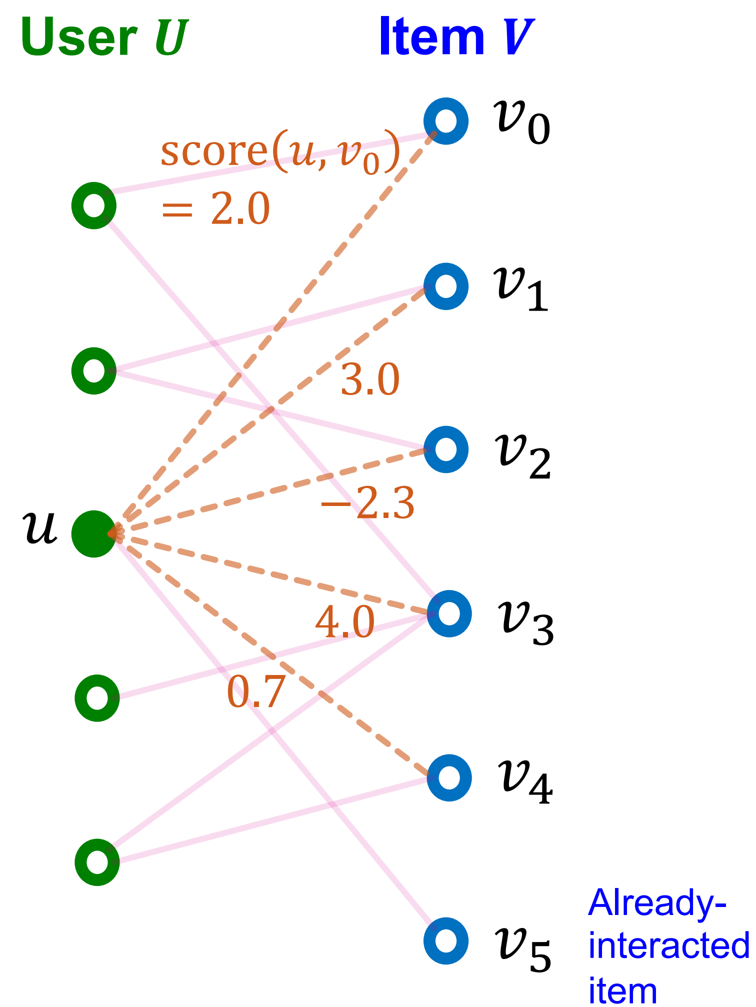
http://cs224w.stanford.edu

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

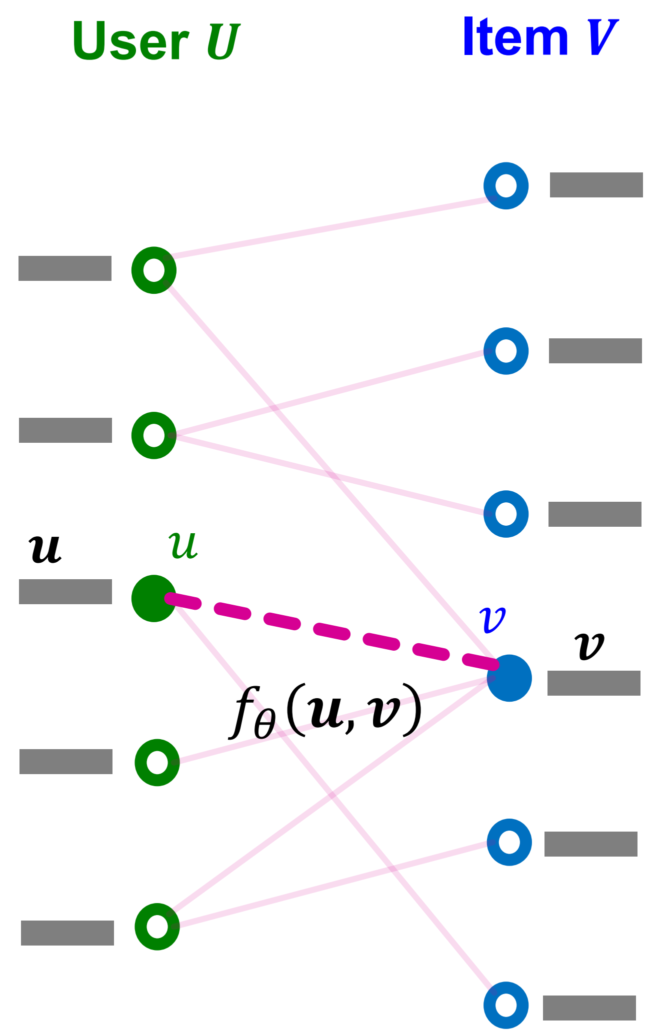
- To get the top-K items, we need a score function for user-item interaction:
  - For $u \in \boldsymbol{U}, v \in \boldsymbol{V}$, we need to get a real-valued scalar **score**(u, v).
  - ***K* items with the largest scores for a given user u** (excluding already-interacted items) are then recommended.



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

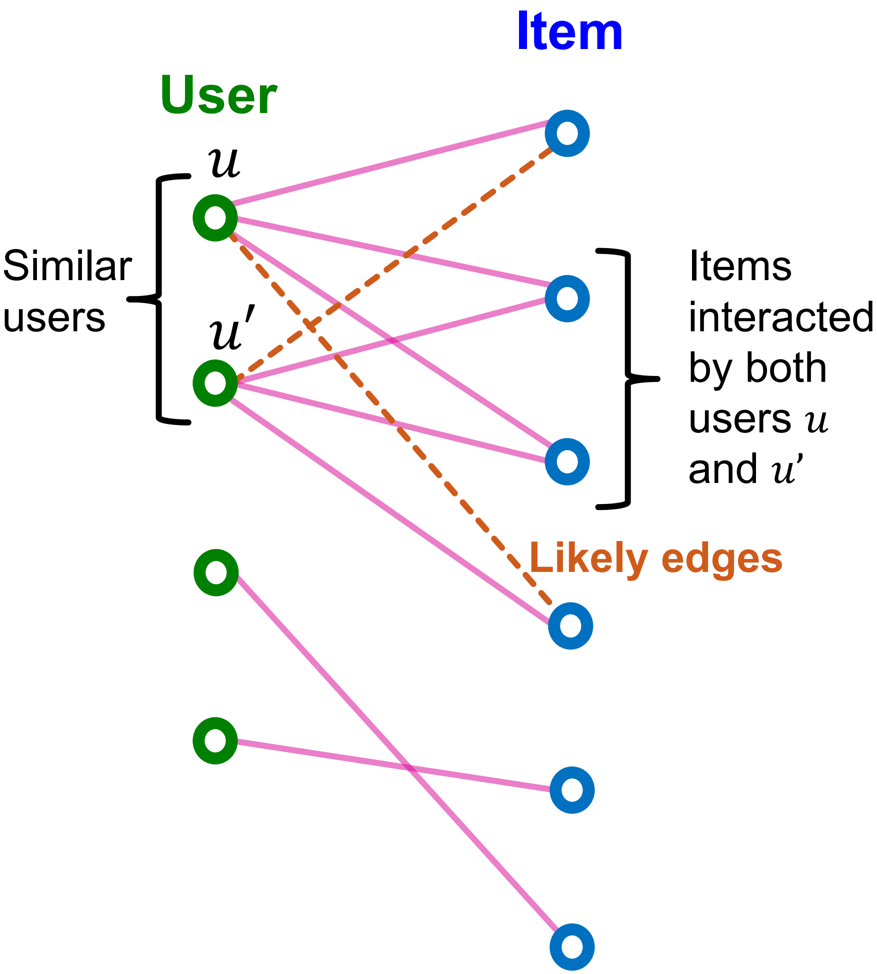
Embedding-Based Models

- 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, $\boxed{\text{score}(u, v) \equiv f_\theta(\boldsymbol{u}, \boldsymbol{v})}$



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.

Stanford CS224W: LightGCN

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

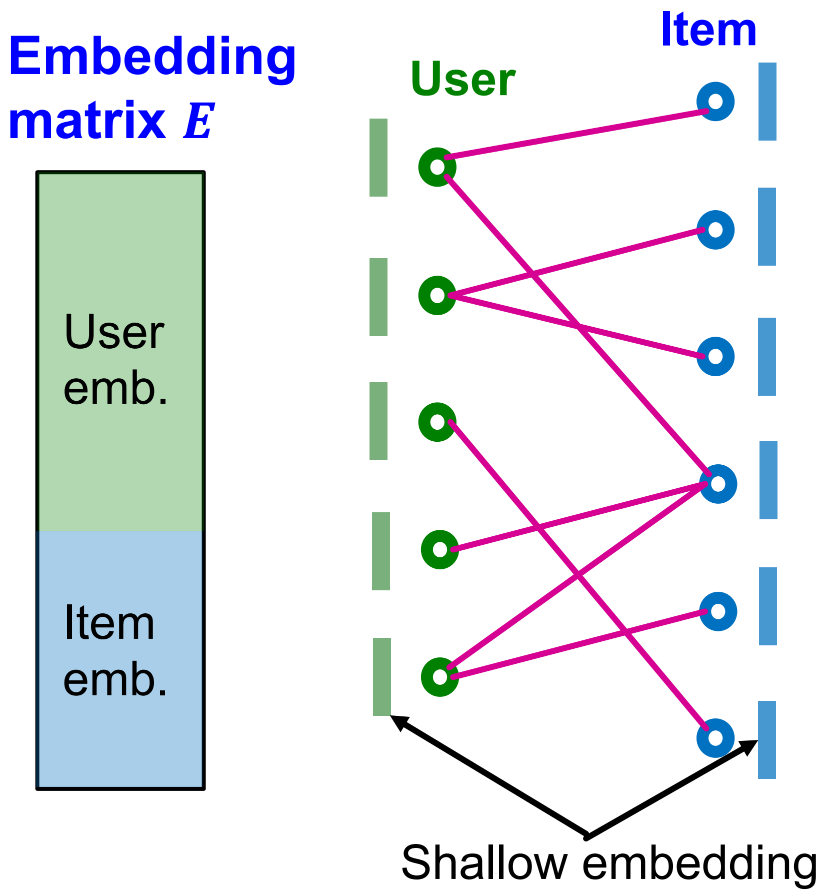
Adjacency and Embedding Matrices

- **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.



Matrix Formulation of GCN

- **Recall**: Diffusion matrix of C&S.
- Let $\boldsymbol{D}$ be the degree matrix of $\boldsymbol{A}$.
- Define the normalized adjacency matrix $\widetilde{\boldsymbol{A}}$ as

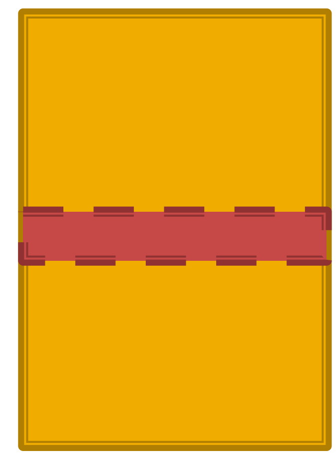
$$\widetilde{\boldsymbol{A}} \equiv \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$$

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:

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

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



Each row stores node embedding

Simplifying GCN (1)

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

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

Original idea from SGC [Wu et al. 2019]

- The final node embedding matrix is given as

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

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

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

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

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

Simplifying GCN (2)

- Removing ReLU significantly simplifies GCN!

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

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

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

- **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

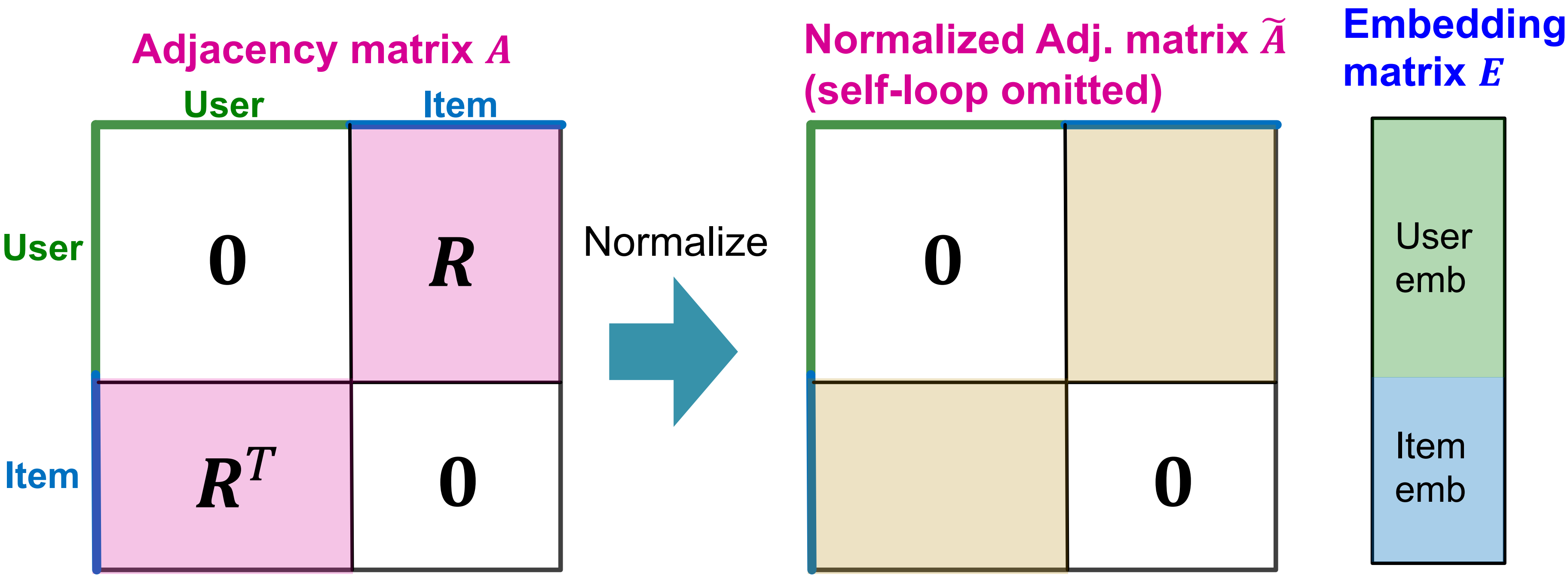
- We can consider **multi-scale diffusion**

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

  - 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, $\alpha_0, \ldots, \alpha_K$, are hyper-parameters.
- For simplicity, LightGCN uses the uniform coefficient, i.e., $\alpha_k = \frac{1}{K+1}$ for $k = 0, \ldots, K$.

LightGCN: Model Overview (1)

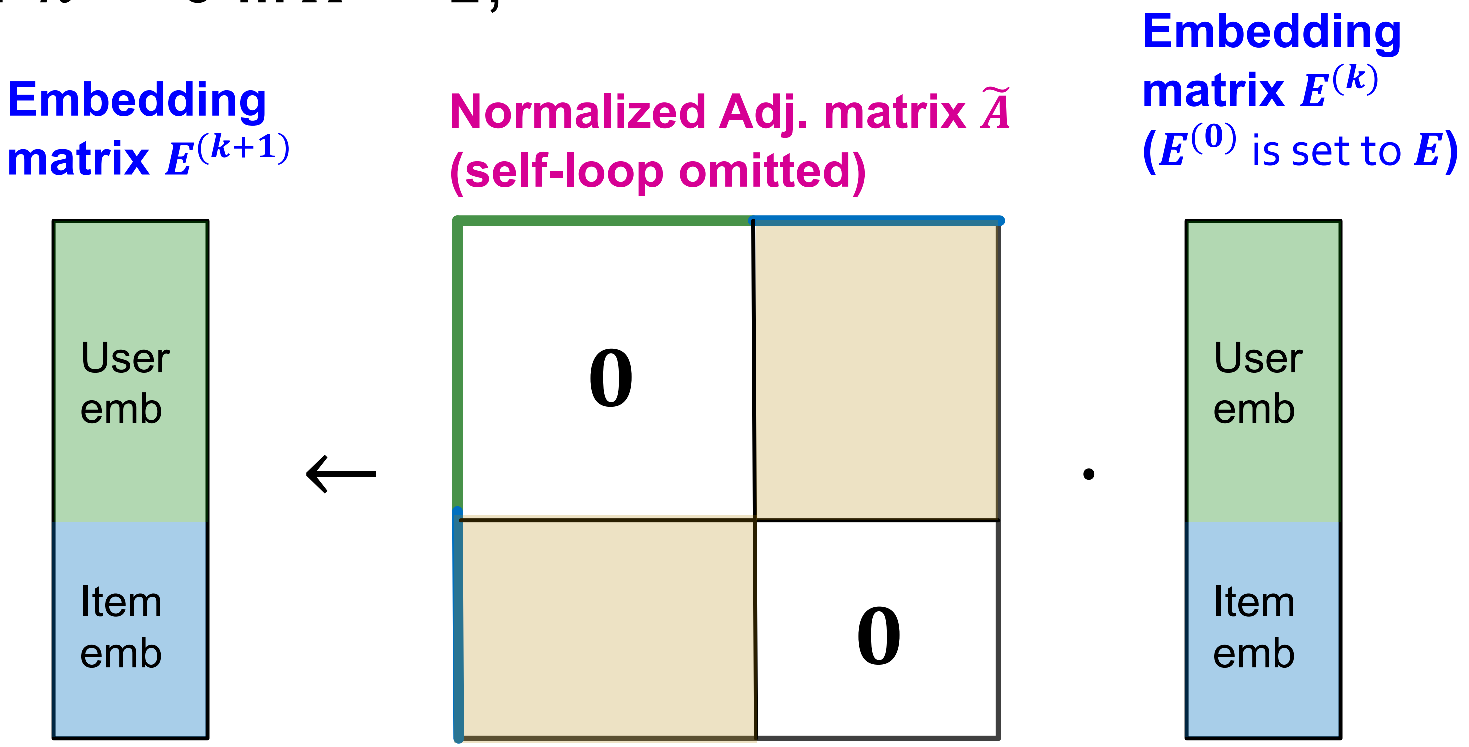
- **Given:**
  - **Adjacency matrix A**
  - **Initial learnable embedding matrix E**



LightGCN: Model Overview (2)

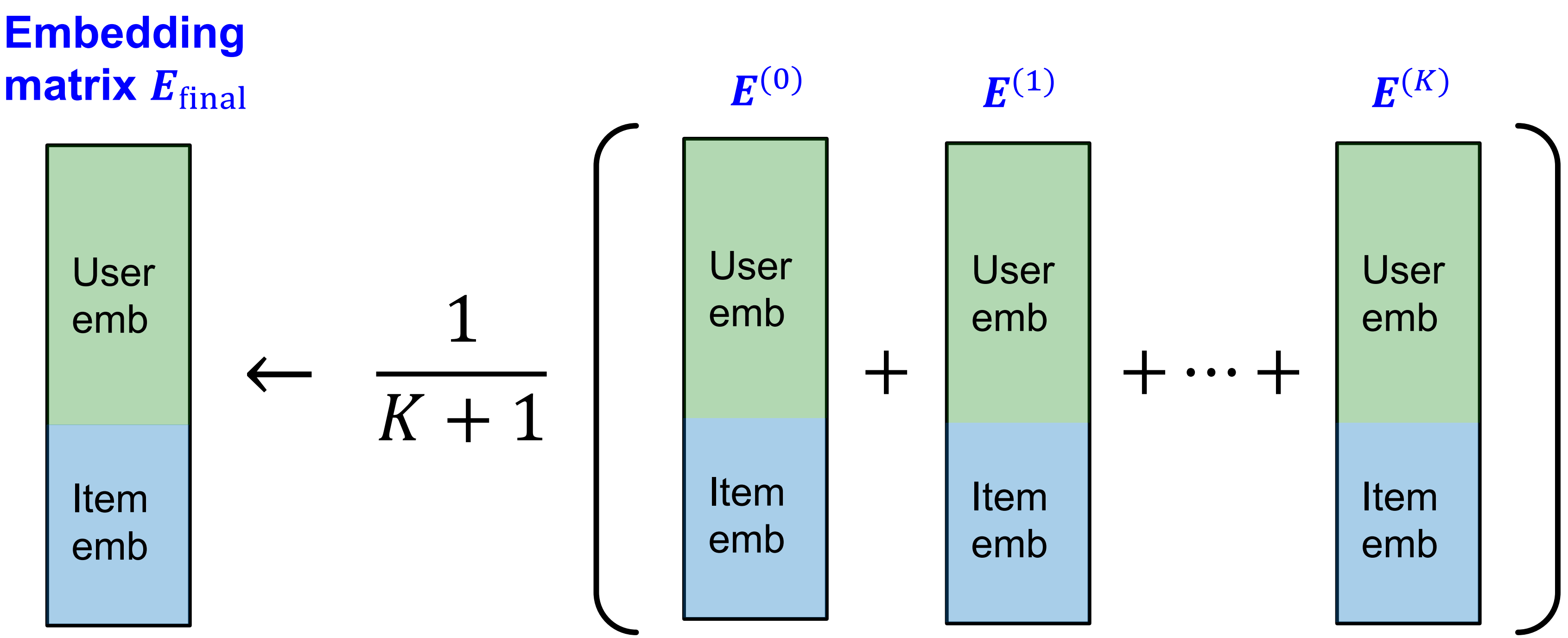
- Iteratively diffuse embedding matrix $\boldsymbol{E}$ using $\widetilde{\boldsymbol{A}}$

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



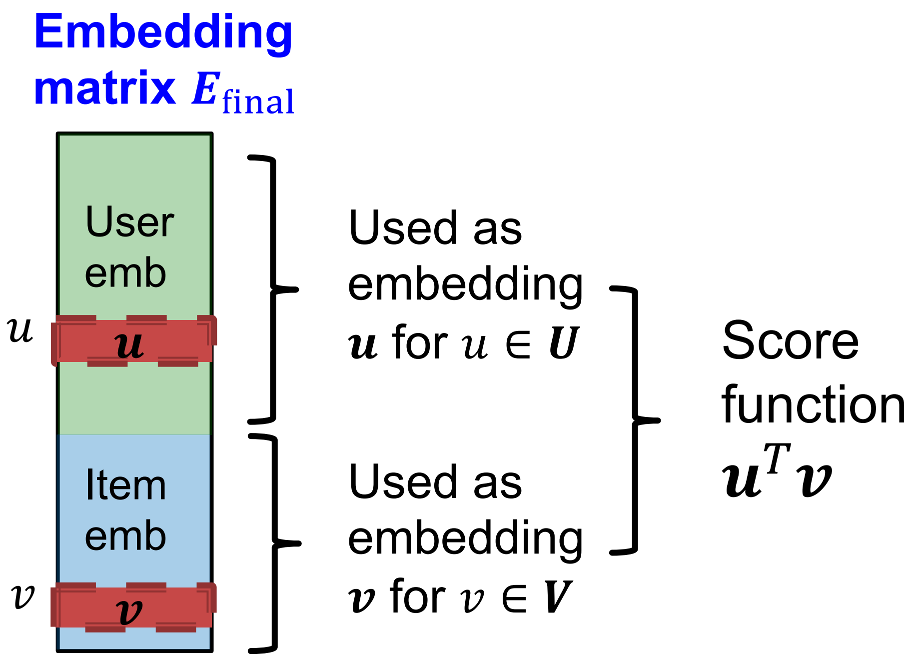
LightGCN: Model Overview (3)

- 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



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)

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

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 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.