



IP PARIS



Recommender Systems

APM_5DS30_TP, Machine Learning with Graphs

Jhony H. Giraldo (Enseignant-Chercheur).

jhony.giraldo@telecom-paris.fr

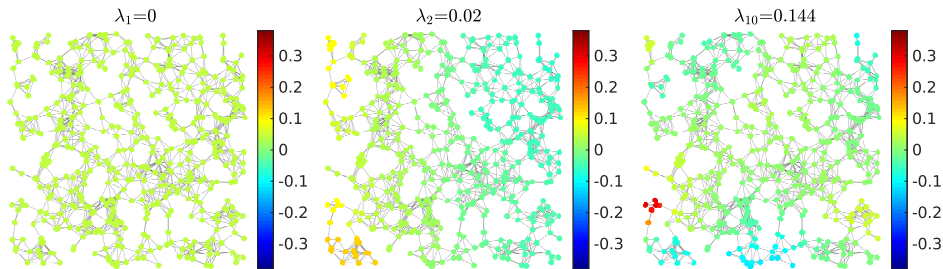
November 21, 2025



Recap

Graph Fourier Transform (GFT)

- ▶ When the graph is undirected, $\mathbf{S}$ is symmetric, and since any real symmetric matrix is guaranteed to have a full set of orthogonal eigenvectors, we have $\mathbf{U}^\top \mathbf{U} = \mathbf{U} \mathbf{U}^\top = \mathbf{I}$
- ▶ In this case, $\mathbf{x} = \mathbf{U} \hat{\mathbf{x}}$ with $\hat{\mathbf{x}} = \mathbf{U}^\top \mathbf{x}$



Eigenvectors/eigenvalues for the Laplacian matrix of a sensor network

- We can define a ChebConv layer as follows:

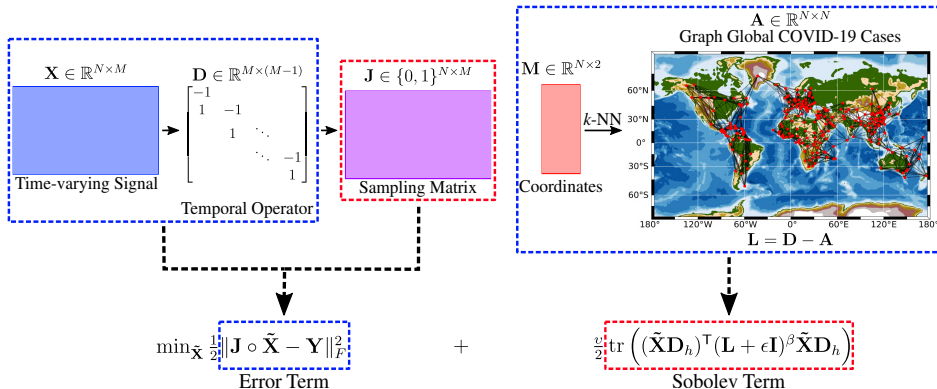
$$\mathbf{H}^{(l+1)} = \sum_{i=1}^k \mathbf{Z}_i \mathbf{W}_i, \quad (1)$$

where $\mathbf{W}_i$ is a linear transformation, $\mathbf{Z}_1 = \mathbf{H}^{(l)}$, $\mathbf{Z}_2 = \tilde{\mathbf{L}}\mathbf{H}^{(l)}$, and $\mathbf{Z}_i = 2\tilde{\mathbf{L}}\mathbf{Z}_{i-1} - \mathbf{Z}_{i-2}$, with $\tilde{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_N} - \mathbf{I}$

- This recursion has been derived using the Chebyshev polynomials, and that's why this model is called ChebConv¹
- If you change the Chebyshev by other families of polynomials, you can get other methods like GegenConv (Gegenbauer polynomials) [Castro-Correa et al.(2024)] or JacobiConv [Wang and Zhang(2022)] (Jacobi polynomials)

Recap

Reconstruction of Time-Varying Graph Signals (Optimization Way)



Recap

Reconstruction of Time-Varying Graph Signals (GNN Way)

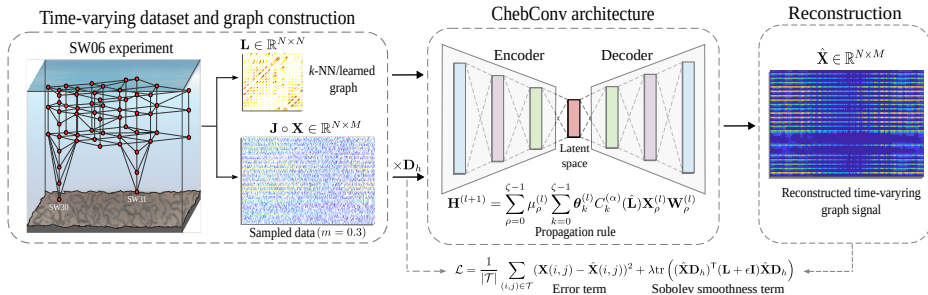


Figure: Source: [Castro-Correa et al.(2024)]

Recap

Forecasting Time-varying Graph Signals

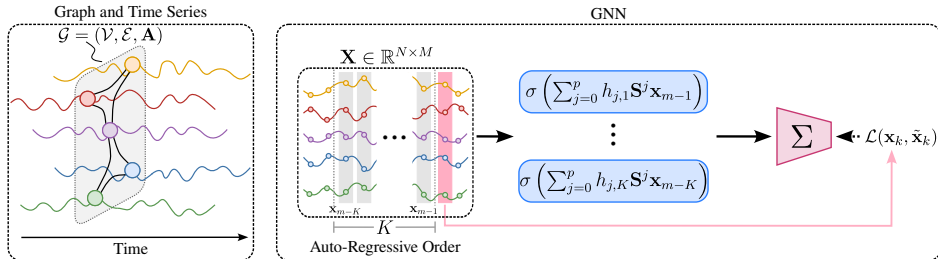


Figure: Forecasting of time-varying graph signals using an MSE loss function using a GNN architecture

Recap GraphCast

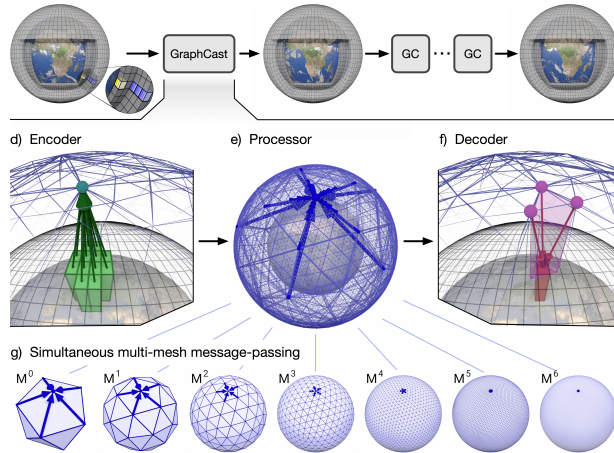


Figure: GraphCast
Source: [Lam et al.(2023)]

Preliminaries of Recommendation

Embedding-Based Models

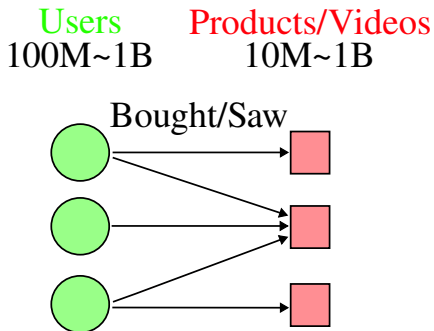
Neural Graph Collaborative Filtering (NGCF)

LightGCN

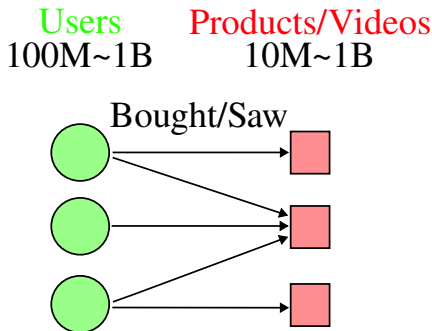
Preliminaries of Recommendation

► Information Explosion in the Era of the Internet

- 10K+ movies in Netflix
- 12M products in Amazon (350M on Marketplace)
- 70M+ music tracks on Spotify
- 10B+ videos on YouTube
- 200B+ pins (images) on Pinterest



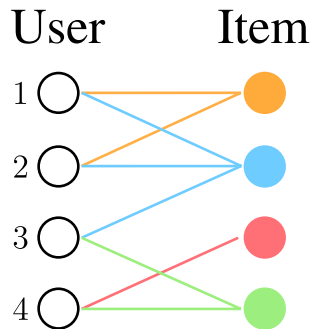
- ▶ Information Explosion in the Era of the Internet
 - ▶ 10K+ movies in Netflix
 - ▶ 12M products in Amazon (350M on Marketplace)
 - ▶ 70M+ music tracks on Spotify
 - ▶ 10B+ videos on YouTube
 - ▶ 200B+ pins (images) on Pinterest
- ▶ **Personalized recommendation (*i.e.*, suggesting a small number of interesting items for each user)** is critical for users to effectively explore the content of their interest



Preliminaries of Recommendation

Recommender Systems as a Graph

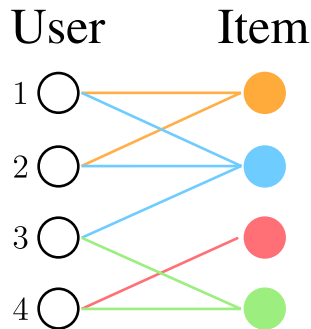
- Recommender system can be naturally modeled as a bipartite graph



Preliminaries of Recommendation

Recommender Systems as a Graph

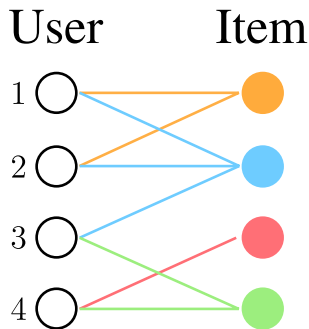
- ▶ Recommender system can be naturally modeled as a bipartite graph
- ▶ A graph with two node types: **users** and **items**



Preliminaries of Recommendation

Recommender Systems as a Graph

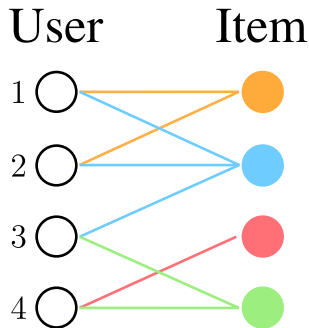
- ▶ Recommender system can be naturally modeled as a bipartite graph
- ▶ A graph with two node types: **users** and **items**
- ▶ **Edges** connect users and items:



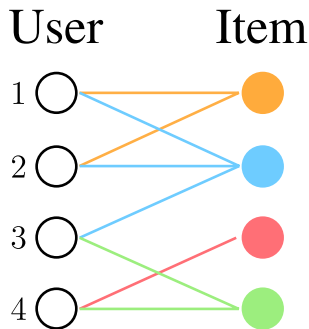
Preliminaries of Recommendation

Recommender Systems 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)



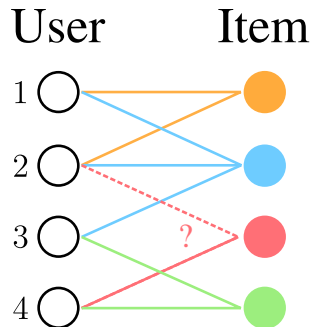
- ▶ Recommender system can be naturally modeled as a bipartite graph
- ▶ A graph with two node types: **users** and **items**
- ▶ **Edges** connect users and items:
 - ▶ Indicates user-item interaction (e.g. click, purchase, review, etc.)
 - ▶ Often associated with timestamp (timing of the interaction)
- ▶ The bipartite representation is a good first exploration, however, real systems are more complex than that, e.g. we have knowledge graphs in the items



Preliminaries of Recommendation

Recommendation Task

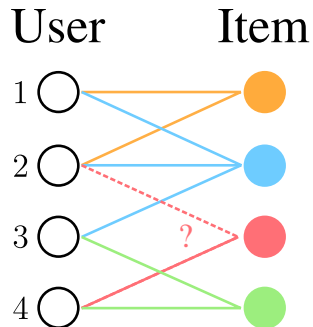
- Input: Past **user-item interactions**



Preliminaries of Recommendation

Recommendation Task

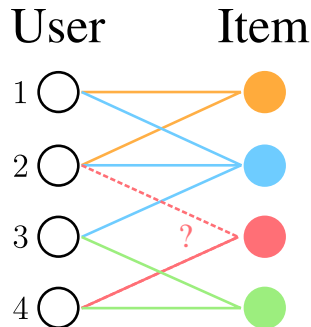
- ▶ Input: Past **user-item interactions**
- ▶ Task: Predict new items each user will interact with in the future



Preliminaries of Recommendation

Recommendation Task

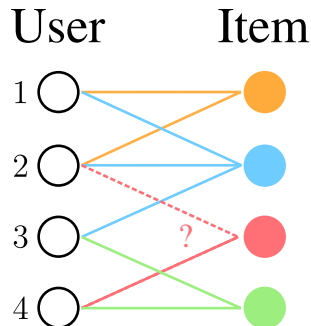
- ▶ Input: Past **user-item interactions**
- ▶ Task: Predict new items each user will interact with in the future
- ▶ Can be cast as a **link prediction** problem
 - ▶ Predict new user-item interaction edges given the past edges.



Preliminaries of Recommendation

Recommendation Task

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



Preliminaries of Recommendation

Modern Recommender Systems

- ▶ We cannot evaluate $f(u, v)$ for every user u – item v pair (e.g. $f(u, v) = \langle \mathbf{z}_u, \mathbf{z}_v \rangle$)
- ▶ Solution, 2-stage process: (i) Candidate generation (cheap, fast), and (ii) ranking (slow, accurate)

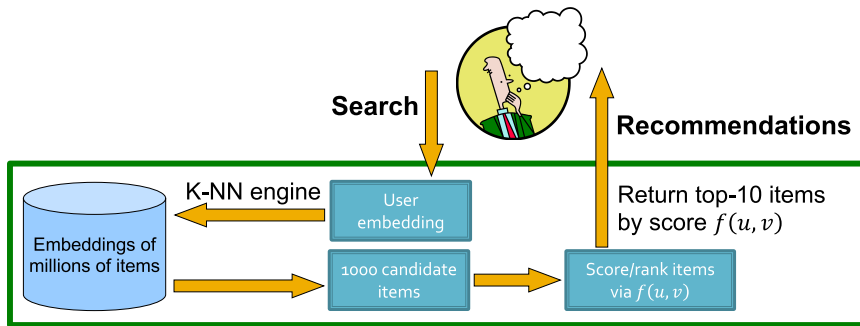


Figure: Source: CS224W, Machine Learning with Graphs, Stanford

Preliminaries of Recommendation

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

Preliminaries of Recommendation

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 are items that the user will interact with in the future

Preliminaries of Recommendation

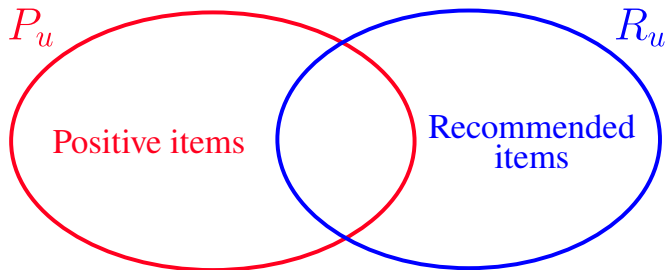
Top- k Recommendation

- ▶ 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 are items that the user will interact with in the future
- ▶ How do we define an evaluation metric? $\text{Recall}@k$

Preliminaries of Recommendation

Evaluation Metric: Recall@ k

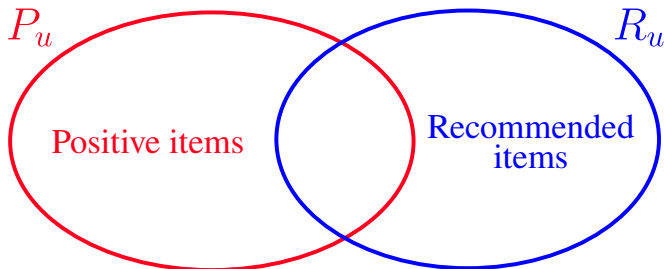
- For each user u :
 - Let P_u be a set of positive items the user will interact in the future
 - Let R_u be a set of items recommended by the model



Preliminaries of Recommendation

Evaluation Metric: Recall@ k

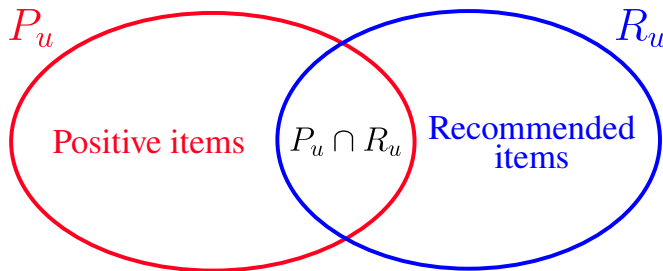
- ▶ For each user u :
 - ▶ Let P_u be a set of positive items the user will interact in the future
 - ▶ Let R_u be a set of items recommended by the model
- ▶ In top- k recommendation, $|R_u| = k$
- ▶ Items that the user has already interacted are excluded from R_u



Preliminaries of Recommendation

Evaluation Metric: Recall@ k

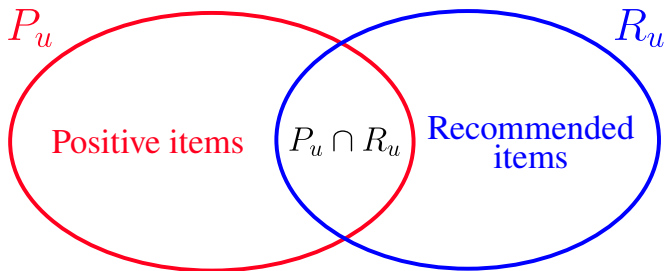
- Recall@ k for user u is $|P_u \cap R_u|/|P_u|$
 - Higher value indicates more positive items are recommended in top- k for user u



Preliminaries of Recommendation

Evaluation Metric: Recall@ k

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



- ▶ The final Recall@ k is computed by averaging the recall values across all users

Preliminaries of Recommendation

Societal Risks

Preliminaries of Recommendation

Societal Risks

- ▶ **Question:** How many of you have opened TikTok/Instagram/YouTube "just for 5 minutes" and realized 30 minutes disappeared?

- ▶ **Question:** How many of you have opened TikTok/Instagram/YouTube "just for 5 minutes" and realized 30 minutes disappeared?
- ▶ This is not accidental; it is a direct consequence of the objective functions we'll discuss
- ▶ Recommender systems optimize engagement-based metrics (clicks, watch time, etc)
- ▶ Recall@ k gives the user what they are most likely to interact with
- ▶ If the user has a history of being drawn to certain content (even if it's harmful/addictive), the system will amplify it

- ▶ **Question:** How many of you have opened TikTok/Instagram/YouTube "just for 5 minutes" and realized 30 minutes disappeared?
- ▶ This is not accidental; it is a direct consequence of the objective functions we'll discuss
- ▶ Recommender systems optimize engagement-based metrics (clicks, watch time, etc)
- ▶ Recall@ k gives the user what they are most likely to interact with
- ▶ If the user has a history of being drawn to certain content (even if it's harmful/addictive), the system will amplify it
- ▶ Going beyond this is outside the scope of the course — but there are active research fields tackling these issues, and it's important to be aware of them

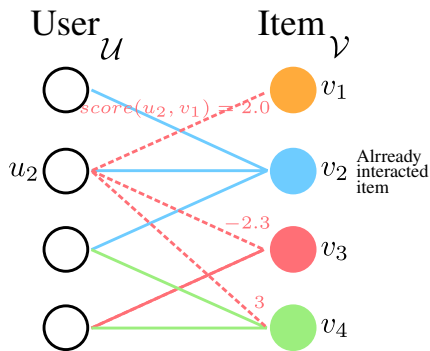
Embedding-Based Models

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

Embedding-Based Models

Score Function

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

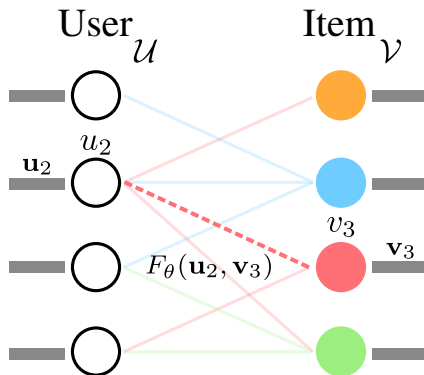


For $k = 2$, recommended items for user u_2 would be $\{v_1, v_4\}$

Embedding-Based Models

Models

- ▶ We consider embedding-based models for scoring user-item interactions
- ▶ For each user $u \in \mathcal{U}$, let $\mathbf{u} \in \mathbb{R}^D$ be its D -dimensional embedding
- ▶ For each item $v \in \mathcal{V}$, 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, $score(u, v) := F_\theta(\mathbf{u}, \mathbf{v})$



- ▶ Embedding-based models have three kinds of parameters:
 - ▶ An encoder to generate user embeddings $\{\mathbf{u}\}_{u \in \mathcal{U}}$
 - ▶ An encoder to generate item embeddings $\{\mathbf{v}\}_{v \in \mathcal{V}}$
 - ▶ Score function $F_{\theta}(\cdot, \cdot)$

Embedding-Based Models

Training Objectives

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

Embedding-Based Models

Surrogate Loss Functions

- Why can we not just use $\text{recall}@k$ to train some deep learning model?

Embedding-Based Models

Surrogate Loss Functions

- ▶ Why can we not just use $\text{recall}@k$ to train some deep learning model?
- ▶ It's not differentiable, then we cannot apply efficient gradient-based optimization.

- ▶ Why can we not just use $\text{recall}@k$ to train some deep learning model?
- ▶ It's not differentiable, then we 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

Embedding-Based Models

Binary Loss

- ▶ Define positive/negative edges
 - ▶ A set of positive edges $\mathcal{E}$ (i.e., observed/training user-item interactions)
 - ▶ A set of negative edges $\mathcal{E}_{\text{neg}} = \{(u, v) \mid (u, v) \notin \mathcal{E}, u \in \mathcal{U}, v \in \mathcal{V}\}$

- ▶ Define positive/negative edges
 - ▶ A set of positive edges $\mathcal{E}$ (i.e., observed/training user-item interactions)
 - ▶ A set of negative edges $\mathcal{E}_{\text{neg}} = \{(u, v) \mid (u, v) \notin \mathcal{E}, u \in \mathcal{U}, v \in \mathcal{V}\}$
- ▶ Binary loss: Binary classification of positive/negative edges using $\sigma(F_{\theta}(\mathbf{u}, \mathbf{v}))$:

$$-\frac{1}{|\mathcal{E}|} \sum_{(u,v) \in \mathcal{E}} \log(\sigma(F_{\theta}(\mathbf{u}, \mathbf{v}))) - \frac{1}{|\mathcal{E}_{\text{neg}}|} \sum_{(u,v) \in \mathcal{E}_{\text{neg}}} \log(1 - \sigma(F_{\theta}(\mathbf{u}, \mathbf{v}))),$$

where $\sigma(\cdot)$ is the sigmoid function

Embedding-Based Models

Binary Loss

- ▶ Define positive/negative edges
 - ▶ A set of positive edges $\mathcal{E}$ (i.e., observed/training user-item interactions)
 - ▶ A set of negative edges $\mathcal{E}_{\text{neg}} = \{(u, v) \mid (u, v) \notin \mathcal{E}, u \in \mathcal{U}, v \in \mathcal{V}\}$
- ▶ Binary loss: Binary classification of positive/negative edges using $\sigma(F_{\theta}(\mathbf{u}, \mathbf{v}))$:

$$-\frac{1}{|\mathcal{E}|} \sum_{(u,v) \in \mathcal{E}} \log(\sigma(F_{\theta}(\mathbf{u}, \mathbf{v}))) - \frac{1}{|\mathcal{E}_{\text{neg}}|} \sum_{(u,v) \in \mathcal{E}_{\text{neg}}} \log(1 - \sigma(F_{\theta}(\mathbf{u}, \mathbf{v}))),$$

where $\sigma(\cdot)$ is the sigmoid function

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

Embedding-Based Models

Issue with binary loss

- ▶ 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.

Embedding-Based Models

Issue with binary loss

- ▶ 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?

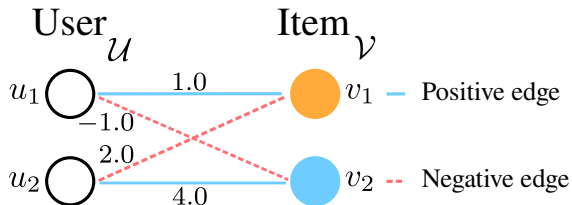
Embedding-Based Models

Issue with binary loss

- ▶ 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?
- ▶ Let's see an example

Embedding-Based Models

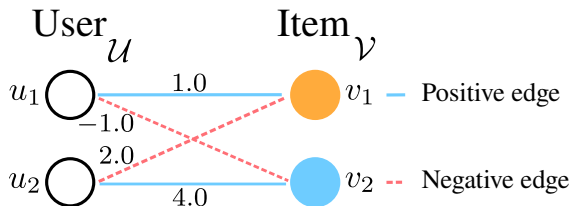
Issue with binary loss



- ▶ Let's consider the simplest case: We have two users, two items
 - ▶ Metric: Recall@1
 - ▶ A model assigns the score for every user-item pair (as shown above)

Embedding-Based Models

Issue with binary loss



- ▶ Let's consider the simplest case: We have two users, two items
 - ▶ Metric: Recall@1
 - ▶ A model assigns the score for every user-item pair (as shown above)
- ▶ Training Recall@1 is 1.0 (perfect score), because v_1 is correctly recommended to u_1 , and the same is true for v_2
- ▶ However, the binary loss would still penalize the model prediction because the negative (u_2, v_1) edge gets the higher score than the positive edge (u_1, v_1)

Embedding-Based Models

Issue with binary loss

► What do we learn from this?

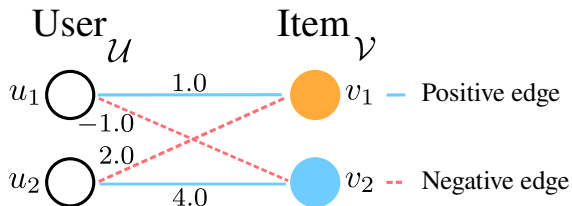
Embedding-Based Models

Issue with binary loss

- ▶ What do we learn from this?
- ▶ 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, *i.e.*, defined for each user
- ▶ Therefore, the *non-personalized* binary loss is overly-strict for the personalized recall metric

Embedding-Based Models

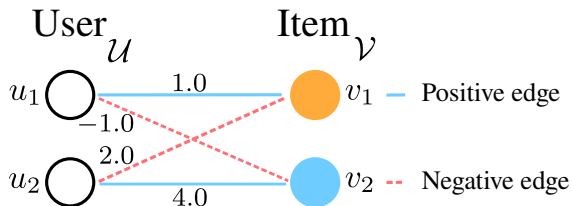
Desirable surrogate loss



- What did we learn? We need a surrogate loss function 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 don't care about the score ordering across users

Embedding-Based Models

Desirable surrogate loss

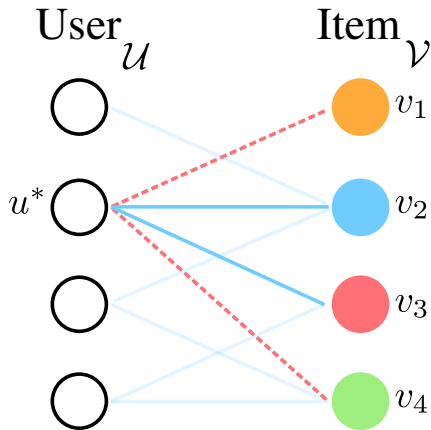


- What did we learn? We need a surrogate loss function defined in a *personalized* manner
 - For each user, we want the scores of positive items to be higher than those of the negative items
 - We don't care about the score ordering across users
- A loss function called *Bayesian Personalized Ranking* (BPR) loss achieves this!

Embedding-Based Models

Loss function: BPR loss

- ▶ Bayesian Personalized Ranking (BPR) loss is a personalized surrogate loss that aligns better with the Recall@K metric
- ▶ For each user $u^* \in \mathcal{U}$, define the rooted positive/negative edges as:



Embedding-Based Models

Loss function: BPR loss

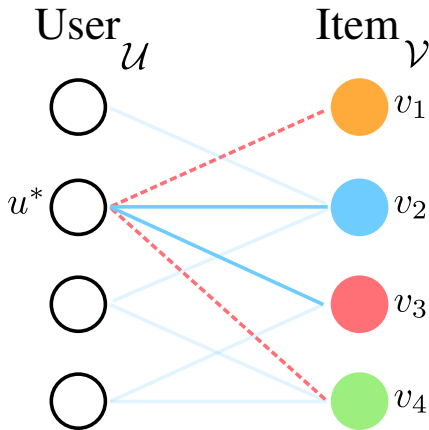
- ▶ Bayesian Personalized Ranking (BPR) loss is a personalized surrogate loss that aligns better with the Recall@K metric
- ▶ For each user $u^* \in \mathcal{U}$, define the rooted positive/negative edges as:

- ▶ Positive edges rooted at u^*

$$\mathcal{E}(u^*) := \{(u^*, v) \mid (u^*, v) \in \mathcal{E}\}$$

- ▶ Negative edges rooted at u^*

$$\mathcal{E}_{\text{neg}}(u^*) := \{(u^*, v) \mid (u^*, v) \in \mathcal{E}_{\text{neg}}\}$$



Embedding-Based Models

Loss function: BPR loss

- ▶ Training objective: For each user u^* , we want the scores of rooted positive edges $\mathcal{E}(u^*)$ to be higher than those of rooted negative edges $\mathcal{E}_{\text{neg}}(u^*)$
 - ▶ It aligns with the personalized nature of the recall metric

Embedding-Based Models

Loss function: BPR loss

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

$$\text{Loss}(u^*) = \frac{1}{|\mathcal{E}(u^*)||\mathcal{E}_{\text{neg}}(u^*)|} \sum_{(u^*, v_{\text{pos}}) \in \mathcal{E}(u^*)} \sum_{(u^*, v_{\text{neg}}) \in \mathcal{E}_{\text{neg}}(u^*)} -\log \left(\sigma \left(F_{\theta}(u^*, v_{\text{pos}}) - F_{\theta}(u^*, v_{\text{neg}}) \right) \right),$$

where the function $F_{\theta}(u^*, v_{\text{pos}}) - F_{\theta}(u^*, v_{\text{neg}})$ is encouraged to be positive for each user, *i.e.*, positive edge scores are higher than negative edge scores

- ▶ The BPR loss can be approximated using a mini-batch

Embedding-Based Models

Loss function: BPR loss

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

$$\text{Loss}(u^*) = \frac{1}{|\mathcal{E}(u^*)||\mathcal{E}_{\text{neg}}(u^*)|} \sum_{(u^*, v_{\text{pos}}) \in \mathcal{E}(u^*)} \sum_{(u^*, v_{\text{neg}}) \in \mathcal{E}_{\text{neg}}(u^*)} -\log \left(\sigma \left(F_{\theta}(u^*, v_{\text{pos}}) - F_{\theta}(u^*, v_{\text{neg}}) \right) \right),$$

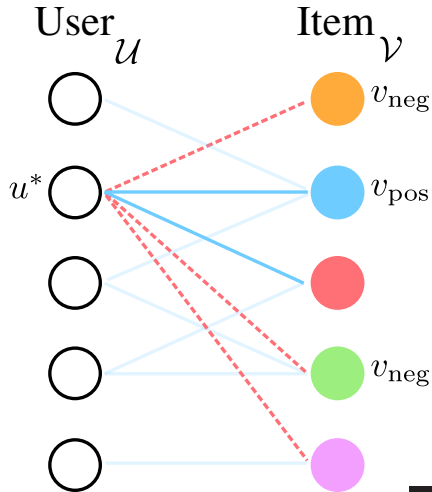
where the function $F_{\theta}(u^*, v_{\text{pos}}) - F_{\theta}(u^*, v_{\text{neg}})$ is encouraged to be positive for each user, *i.e.*, positive edge scores are higher than negative edge scores

- ▶ The BPR loss can be approximated using a mini-batch
- ▶ The final BPR loss is given by $\frac{1}{|\mathcal{U}|} \sum_{u^* \in \mathcal{U}} \text{Loss}(u^*)$

Embedding-Based Models

Loss function: BPR loss

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

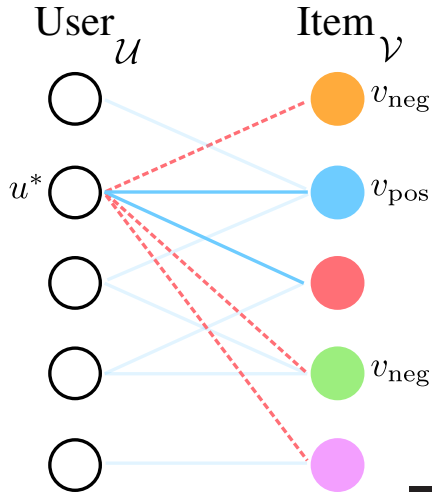


Embedding-Based Models

Loss function: BPR loss

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

$$\frac{1}{|\mathcal{U}_{\text{mini}}|} \sum_{u^* \in \mathcal{U}_{\text{mini}}} \frac{1}{|\mathcal{V}_{\text{neg}}|} \sum_{v_{\text{neg}} \in \mathcal{V}_{\text{neg}}} -\log \left(\sigma \left(F_{\theta}(u^*, v_{\text{pos}}) - F_{\theta}(u^*, v_{\text{neg}}) \right) \right)$$



Embedding-Based Models

Summary: Embedding-based models

- ▶ We have introduced:
 - ▶ Recall@K as a metric for personalized recommendation
 - ▶ Embedding-based models with three 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 achieve state-of-the-art performance in recommender systems

Embedding-Based Models

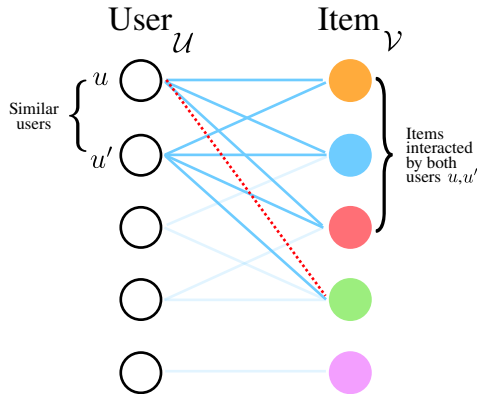
Summary: Embedding-based models

- ▶ We have introduced:
 - ▶ Recall@K as a metric for personalized recommendation
 - ▶ Embedding-based models with three 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 achieve state-of-the-art performance in recommender systems
- ▶ Why do they work so well?

Embedding-Based Models

Why do embedding-based models work?

- ▶ Main idea: Collaborative filtering
 - ▶ Recommend items for a user by collecting preferences of many other similar users
 - ▶ Similar users tend to prefer similar items
- ▶ Question: How to capture similarity between users/items?



Embedding-Based Models

Why do embedding-based 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

Embedding-Based Models

GNNs for recommender systems

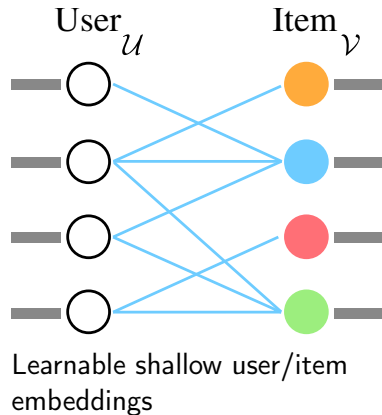
- ▶ We'll see two representative GNN approaches for recommender systems
- ▶ Neural Graph Collaborative Filtering (NGCF) [Wang et al.(2019)]
- ▶ LightGCN [He et al.(2020)]
 - ▶ Graph-based approaches improve the conventional collaborative filtering models (*i.e.*, shallow encoders) by explicitly modeling graph structure using GNNs
 - ▶ They don't assume user/item features

Neural Graph Collaborative Filtering (NGCF)

Neural Graph Collaborative Filtering (NGCF)

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 $u \in \mathcal{U}$ and $v \in \mathcal{V}$, we prepare shallow learnable embeddings
 $\mathbf{u}, \mathbf{v} \in \mathbb{R}^D$
 - ▶ The score function for user u and item v is $f(u, v) := \langle \mathbf{z}_u, \mathbf{z}_v \rangle$



Neural Graph Collaborative Filtering (NGCF)

What are the limitation of shallow encoders?

Neural Graph Collaborative Filtering (NGCF)

What are the limitation of shallow encoders?

- ▶ The model itself does not explicitly capture graph structure
- ▶ The graph structure is only implicitly captured in the training objective

Neural Graph Collaborative Filtering (NGCF)

What are the limitation 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

Neural Graph Collaborative Filtering (NGCF)

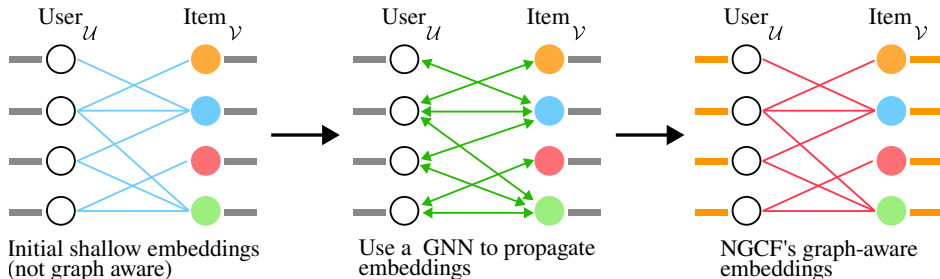
NGCF Motivation

- ▶ 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

Neural Graph Collaborative Filtering (NGCF)

NGCF Overview

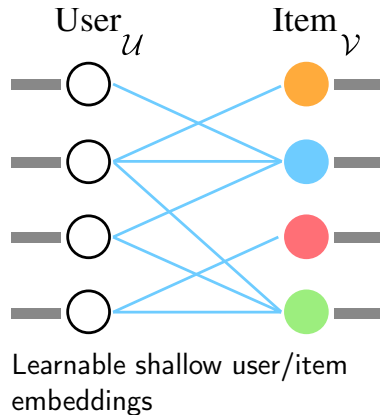
- ▶ Neural Graph Collaborative Filtering (NGCF) explicitly incorporates high-order graph structure when generating user/item embeddings
- ▶ Key idea: Use a GNN to generate graph-aware user/item embeddings



Neural Graph Collaborative Filtering (NGCF)

NGCF framework

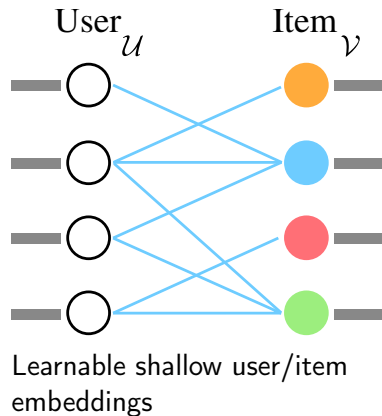
- ▶ Given the user-item bipartite graph:
 - ▶ Prepare shallow learnable embedding for each node
 - ▶ Use multi-layer GNNs to propagate embeddings along the bipartite graph
 - ▶ High-order graph structure is captured
 - ▶ Final embeddings are explicitly graph-aware
- ▶ Two kinds of learnable parameters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



Neural Graph Collaborative Filtering (NGCF)

Initial node embeddings

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



Neural Graph Collaborative Filtering (NGCF)

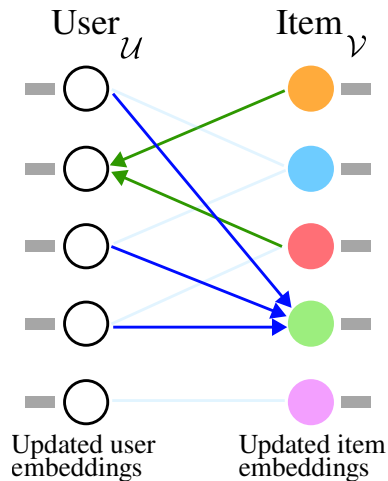
Neighbor aggregation

- Iteratively update node embeddings using neighboring embeddings

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

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

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



Neural Graph Collaborative Filtering (NGCF)

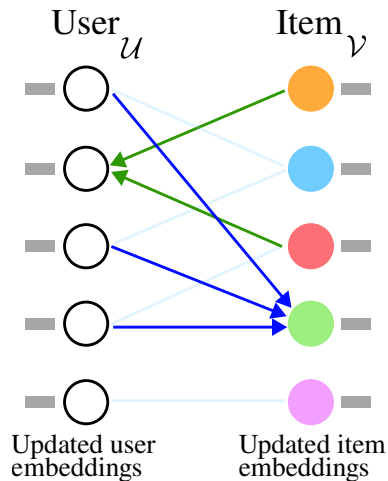
Neighbor aggregation

- Iteratively update node embeddings using neighboring embeddings

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

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

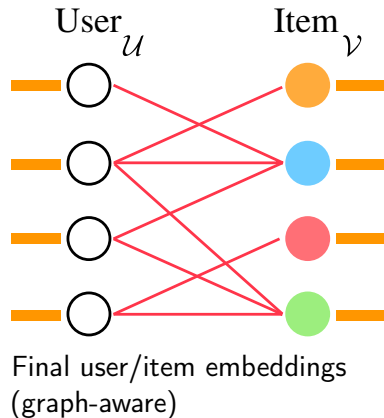
- High-order graph structure is captured through iterative neighbor aggregation
- Different architecture choices are possible for AGGR and COMBINE:
 - AGGR($\cdot$) can be MEAN($\cdot$)
 - COMBINE($\mathbf{x}, \mathbf{y}$) can be $\text{ReLU}(\text{Linear}(\text{Concat}(\mathbf{x}, \mathbf{y})))$



Neural Graph Collaborative Filtering (NGCF)

Final embedding 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 $u \in \mathcal{U}$, $v \in \mathcal{V}$, we set $\mathbf{u} \leftarrow \mathbf{h}_u^{(K)}$, $\mathbf{v} \leftarrow \mathbf{h}_v^{(K)}$
- ▶ Score function is the inner product:
 $score(u, v) = \langle \mathbf{u}, \mathbf{v} \rangle$



Neural Graph Collaborative Filtering (NGCF)

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

- ▶ Recall that NGCF jointly learns two kinds of parameters:
 - ▶ Shallow user/item embeddings
 - ▶ GNN's parameters
- ▶ However, 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 dimension)
 - ▶ Shallow embeddings: $\mathcal{O}(ND)$
 - ▶ GNN: $\mathcal{O}(D^2)$
 - ▶ The GNN parameters may not be so essential for performance

LightGCN

LightGCN Motivation

- Can we simplify the GNN used in NGCF (e.g., remove its learnable parameters)?

LightGCN

LightGCN Motivation

- ▶ Can we simplify the GNN used in NGCF (e.g., remove its learnable parameters)?
- ▶ Yes! and the simplification improves the recommendation performance!

LightGCN

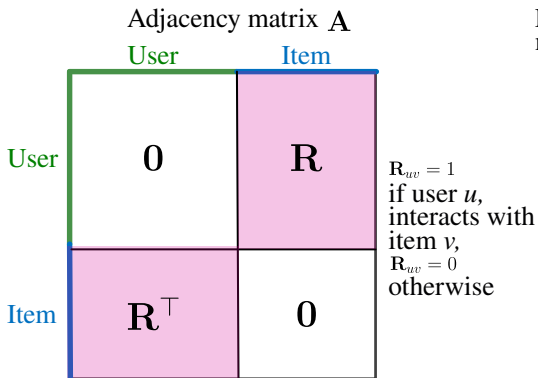
LightGCN Motivation

- ▶ Can we simplify the GNN used in NGCF (e.g., remove its learnable parameters)?
- ▶ Yes! and the 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
 - ▶ It has some similarity with the SGC model [Wu et al.(2019)] we saw and implemented in Lecture 3

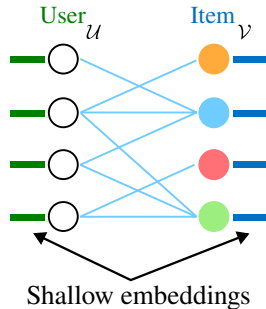
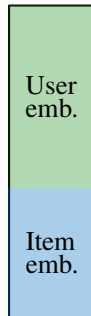
LightGCN

Adjacency and embedding matrices

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



Embedding matrix E



- ▶ Here we use the GCN formulation, but we omit the self-connections, *i.e.*, we normalize the adjacency matrix $\mathbf{A}$ as $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$
- ▶ Let $\mathbf{E}^{(k)}$ be the embedding matrix at k -th layer
- ▶ Therefore, each layer of GCN's aggregation can be written in a matrix form as:

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

- ▶ Here we use the GCN formulation, but we omit the self-connections, *i.e.*, we normalize the adjacency matrix $\mathbf{A}$ as $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$
- ▶ Let $\mathbf{E}^{(k)}$ be the embedding matrix at k -th layer
- ▶ Therefore, each layer of GCN's aggregation can be written in a matrix form as:

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

- ▶ Now, we simplify this model by removing the ReLU nonlinearities, and by invoking our third lecture, we get:

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

- ▶ Here we use the GCN formulation, but we omit the self-connections, *i.e.*, we normalize the adjacency matrix $\mathbf{A}$ as $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$
- ▶ Let $\mathbf{E}^{(k)}$ be the embedding matrix at k -th layer
- ▶ Therefore, each layer of GCN's aggregation can be written in a matrix form as:

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

- ▶ Now, we simplify this model by removing the ReLU nonlinearities, and by invoking our third lecture, we get:

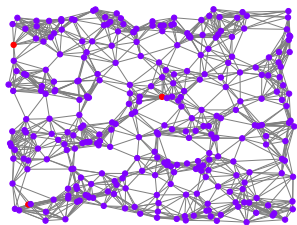
$$\mathbf{E}^{(k)} = \tilde{\mathbf{A}}^k \mathbf{E} \mathbf{W}.$$

- ▶ In LightGCN, we also remove the linear projection, so we end up with $\mathbf{E}^{(k)} = \tilde{\mathbf{A}}^k \mathbf{E}$

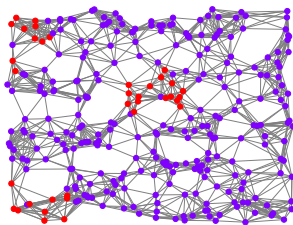
LightGCN

Simplifying GCN

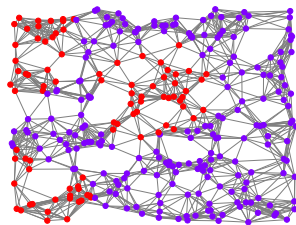
- ▶ $\tilde{\mathbf{A}}^k \mathbf{E}$ diffuses the embeddings along the graph
- ▶ Remember the diffusion sequence in our second lecture, where the shift operator $\mathbf{S}$ is $\tilde{\mathbf{A}}$ in this case



$$\mathbf{x}^{(0)} = \mathbf{x} = \mathbf{S}^0 \mathbf{x}$$



$$\mathbf{x}^{(1)} = \mathbf{S} \mathbf{x}^{(0)} = \mathbf{S}^1 \mathbf{x}$$

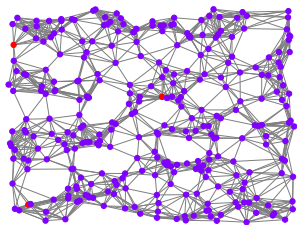


$$\mathbf{x}^{(2)} = \mathbf{S} \mathbf{x}^{(1)} = \mathbf{S}^2 \mathbf{x}$$

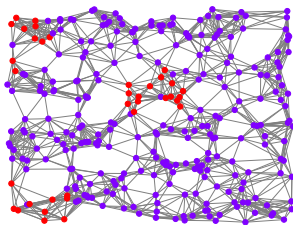
LightGCN

Simplifying GCN

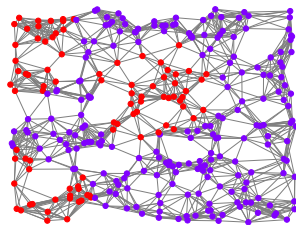
- ▶ $\tilde{\mathbf{A}}^k \mathbf{E}$ diffuses the embeddings along the graph
- ▶ Remember the diffusion sequence in our second lecture, where the shift operator $\mathbf{S}$ is $\tilde{\mathbf{A}}$ in this case



$$\mathbf{x}^{(0)} = \mathbf{x} = \mathbf{S}^0 \mathbf{x}$$



$$\mathbf{x}^{(1)} = \mathbf{S} \mathbf{x}^{(0)} = \mathbf{S}^1 \mathbf{x}$$



$$\mathbf{x}^{(2)} = \mathbf{S} \mathbf{x}^{(1)} = \mathbf{S}^2 \mathbf{x}$$

- ▶ *Algorithm:* Apply $\mathbf{E} \leftarrow \tilde{\mathbf{A}} \mathbf{E}$ for k times, this is the recursive implementation, and you know that $\tilde{\mathbf{A}}^k$ is dense, but it never gets materialized

- An important difference between LightGCN and SGC is the multi-scale diffusion as follows:

$$\alpha_0 \mathbf{E}^{(0)} + \alpha_1 \mathbf{E}^{(1)} + \alpha_2 \mathbf{E}^{(2)} + \dots \alpha_k \mathbf{E}^{(k)} \quad (2)$$

- An important difference between LightGCN and SGC is the multi-scale diffusion as follows:

$$\alpha_0 \mathbf{E}^{(0)} + \alpha_1 \mathbf{E}^{(1)} + \alpha_2 \mathbf{E}^{(2)} + \dots \alpha_k \mathbf{E}^{(k)} \quad (2)$$

- The above definition includes embeddings diffused at multiple hop scales
- Note that $\alpha_0 \mathbf{E}^{(0)} = \alpha_0 \tilde{\mathbf{A}}^0 \mathbf{E}^{(0)}$
- The coefficients $\{\alpha_0, \dots, \alpha_k\}$ are hyperparameters

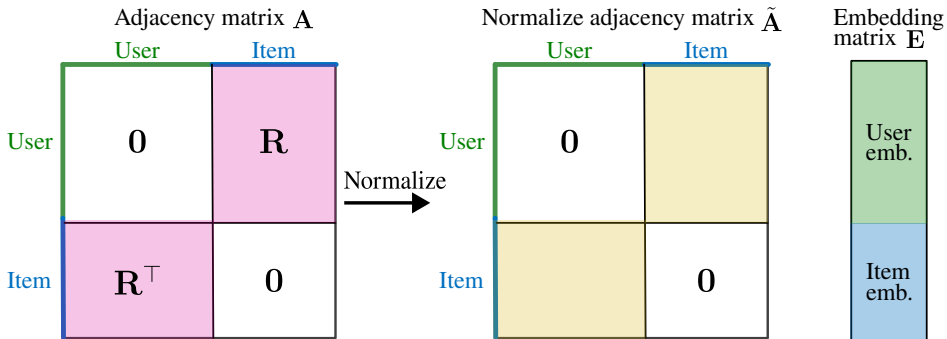
- ▶ An important difference between LightGCN and SGC is the multi-scale diffusion as follows:

$$\alpha_0 \mathbf{E}^{(0)} + \alpha_1 \mathbf{E}^{(1)} + \alpha_2 \mathbf{E}^{(2)} + \dots \alpha_k \mathbf{E}^{(k)} \quad (2)$$

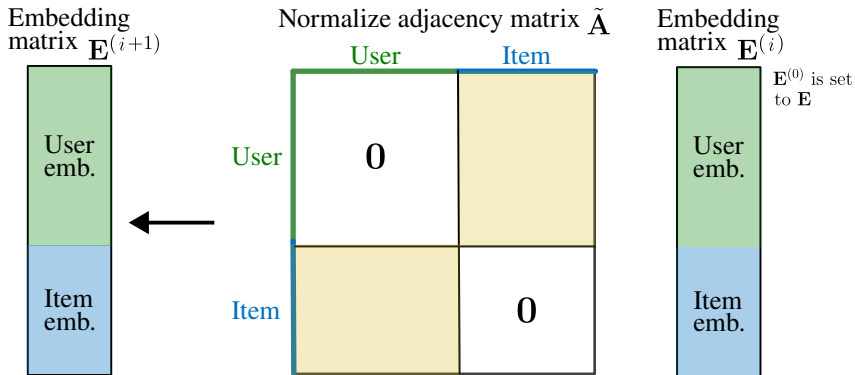
- ▶ The above definition includes embeddings diffused at multiple hop scales
- ▶ Note that $\alpha_0 \mathbf{E}^{(0)} = \alpha_0 \tilde{\mathbf{A}}^0 \mathbf{E}^{(0)}$
- ▶ The coefficients $\{\alpha_0, \dots, \alpha_k\}$ are hyperparameters
- ▶ For simplicity, LightGCN uses uniform coefficients: $\alpha_i = \frac{1}{k+1}$ for $i = 0, \dots, k$

► Given:

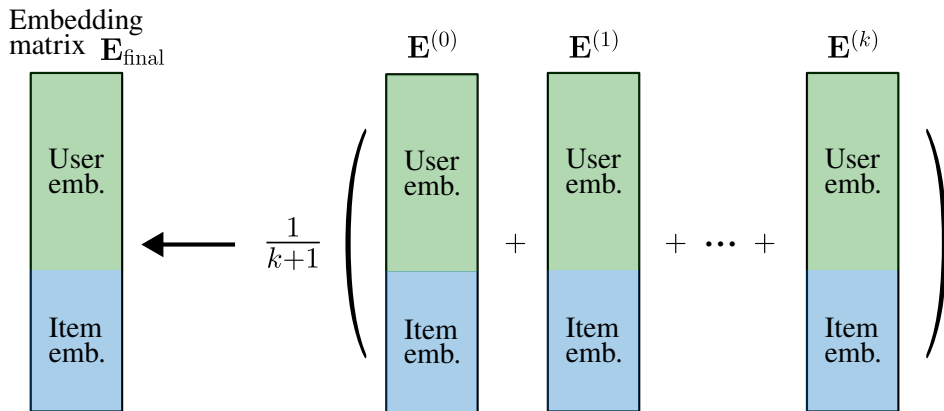
- The adjacency matrix A
- The initial *learnable* embedding matrix E



- ▶ We iteratively diffuse the embedding matrix $\mathbf{E}$ using $\tilde{\mathbf{A}}$
- ▶ For $i = 0, \dots, k - 1$



- Average the embedding matrices at different scales



LightGCN

Overview: LightGCN

- ▶ We use user/item vectors from $\mathbf{E}_{\text{final}}$ as user ($\mathbf{z}_u$ for $u \in \mathcal{U}$) and item ($\mathbf{z}_v$ for $v \in \mathcal{V}$) embeddings
- ▶ We compute the score function as $\mathbf{z}_u^\top \mathbf{z}_v$

Embedding
matrix $\mathbf{E}_{\text{final}}$



LightGCN

Why does it work?

- Why does the simple diffusion propagation work well?

- ▶ Why does the simple diffusion propagation work well?
- ▶ 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

Comparison with shallow encoders

- ▶ 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 is also more computationally expensive due to the additional diffusion step
- ▶ The final embedding of a user/item is obtained by aggregating the embeddings of its multi-hop neighboring nodes

- ▶ 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

Activity

Guided questions

In groups of two or three students, try to answer the following questions:

1. Why is $\text{Recall}@k$ difficult to optimize directly, and how does the BPR loss approximate its behavior?
2. What is the core intuition behind collaborative filtering, and why do shallow embeddings work well?
3. What is the main idea behind NGCF's message passing, and how does it capture higher-order user–item relations?
4. In what ways is LightGCN different from NGCF?
5. What would happen if we increase the number of diffusion layers in LightGCN too much? Why?
6. Could recommender systems be approached differently using large language models? How?

Upload a PDF file with your names and answer to the task “Activity: Guided questions” in Moodle. This activity won't be graded, so no need to use LLMs; take this as an opportunity to reflect on what you learned today

Main takeaways

- ▶ Modern platforms (Netflix, Amazon, Spotify, YouTube, Pinterest) face an information explosion and need personalized recommendations
- ▶ A recommender system can be modeled as a bipartite graph with users and items as nodes and interactions (click, purchase, rating, etc.) as edges
- ▶ The recommendation task can be cast as link prediction: predict which new user–item edges will appear in the future

Main takeaways

- ▶ Modern platforms (Netflix, Amazon, Spotify, YouTube, Pinterest) face an information explosion and need personalized recommendations
- ▶ A recommender system can be modeled as a bipartite graph with users and items as nodes and interactions (click, purchase, rating, etc.) as edges
- ▶ The recommendation task can be cast as link prediction: predict which new user–item edges will appear in the future
- ▶ For each user, we recommend a small set of k items ($k \ll \#items$)
- ▶ We evaluate with $Recall@k$, which measures, per user, the fraction of future positive items that appear in the recommended top- k , then averages over users
- ▶ This metric is inherently personalized (defined per user, not globally)

Main takeaways

- ▶ Each user and item is represented by a low-dimensional embedding
- ▶ A score function (often inner product) predicts how likely a user u will interact with item v
- ▶ The model has three parts: user encoder, item encoder, and score function

Main takeaways

- ▶ Each user and item is represented by a low-dimensional embedding
- ▶ A score function (often inner product) predicts how likely a user u will interact with item v
- ▶ The model has three parts: user encoder, item encoder, and score function
- ▶ Recall@ k is non-differentiable, so we use surrogate loss functions

Main takeaways

- ▶ Each user and item is represented by a low-dimensional embedding
- ▶ A score function (often inner product) predicts how likely a user u will interact with item v
- ▶ The model has three parts: user encoder, item encoder, and score function
- ▶ Recall@ k is non-differentiable, so we use surrogate loss functions
- ▶ NGCF starts with learnable shallow embeddings and applies a GNN over the bipartite graph

Main takeaways

- ▶ Each user and item is represented by a low-dimensional embedding
- ▶ A score function (often inner product) predicts how likely a user u will interact with item v
- ▶ The model has three parts: user encoder, item encoder, and score function
- ▶ Recall@ k is non-differentiable, so we use surrogate loss functions
- ▶ NGCF starts with learnable shallow embeddings and applies a GNN over the bipartite graph
- ▶ LightGCN simplifies GCN since most parameters are in the shallow embeddings
- ▶ Diffusion encourages similar users/items to have similar embeddings because they share many common neighbors.

Main takeaways

- ▶ Each user and item is represented by a low-dimensional embedding
- ▶ A score function (often inner product) predicts how likely a user u will interact with item v
- ▶ The model has three parts: user encoder, item encoder, and score function
- ▶ Recall@ k is non-differentiable, so we use surrogate loss functions
- ▶ NGCF starts with learnable shallow embeddings and applies a GNN over the bipartite graph
- ▶ LightGCN simplifies GCN since most parameters are in the shallow embeddings
- ▶ Diffusion encourages similar users/items to have similar embeddings because they share many common neighbors.
- ▶ LLMs don't outperform GNNs for recommendations (<https://kumo.ai/research/recommendation-systems-llms-graph-transformers/>)

Thank you!

Jhony H. Giraldo: jhony.giraldo@telecom-paris.fr
<https://jhonygiraldo.github.io/>



Jhon A Castro-Correa et al.

Gegenbauer graph neural networks for time-varying signal reconstruction.
IEEE Transactions on Neural Networks and Learning Systems, 2024.



X. He et al.

LightGCN: Simplifying and powering graph convolution network for recommendation.

In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020.



Remi Lam et al.

Learning skillful medium-range global weather forecasting.

Science, 382(6677):1416–1421, 2023.



Xiang Wang et al.

Neural graph collaborative filtering.

In International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 165–174, 2019.



Xiyuan Wang and Muhan Zhang.

How powerful are spectral graph neural networks.

In International Conference on Machine Learning, 2022.



F. Wu et al.

Simplifying graph convolutional networks.

In International Conference on Machine Learning, 2019.