CSCE 670 - Information Storage and Retrieval

Week 11: Deep Learning for Recommendation

Yu Zhang

yuzhang@tamu.edu

Course Website: https://yuzhang-teaching.github.io/CSCE670-S26.html

Adapted from the slides by Prof. James Caverlee

Quiz 3!

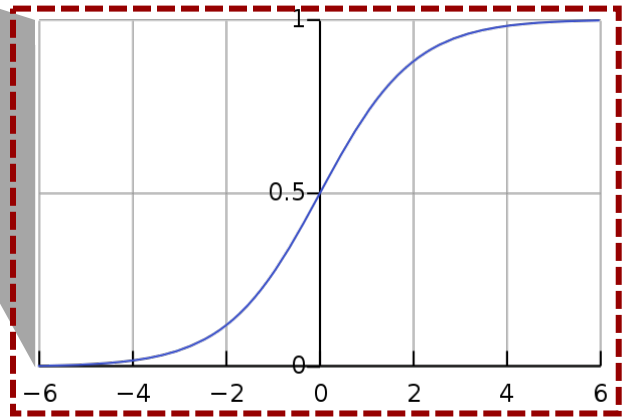
- All policies are the same as Quiz 1 (number of questions, time limit, grading, etc.)
- Scope:
  - Week 8 (word2vec, Neural Ranking)
  - Week 10 (BERT, BERT-Based Ranking)
  - Homework 2
  - Neural Collaborative Filtering / Sequential Recommendation NOT included

Neural Collaborative Filtering

Neural Word Embeddings (word2vec)

[0.3, -0.7]	[-0.9, 0.4]	[0.8, 0.2]		[-0.2, -0.6]	[0.5, 0.9]	[-0.7, -0.1]
openai	*proposes*	*training*	*[MASK]*	*using*	*reinforcement*	*learning*
			↓			
			chatgpt			

Nonlinear



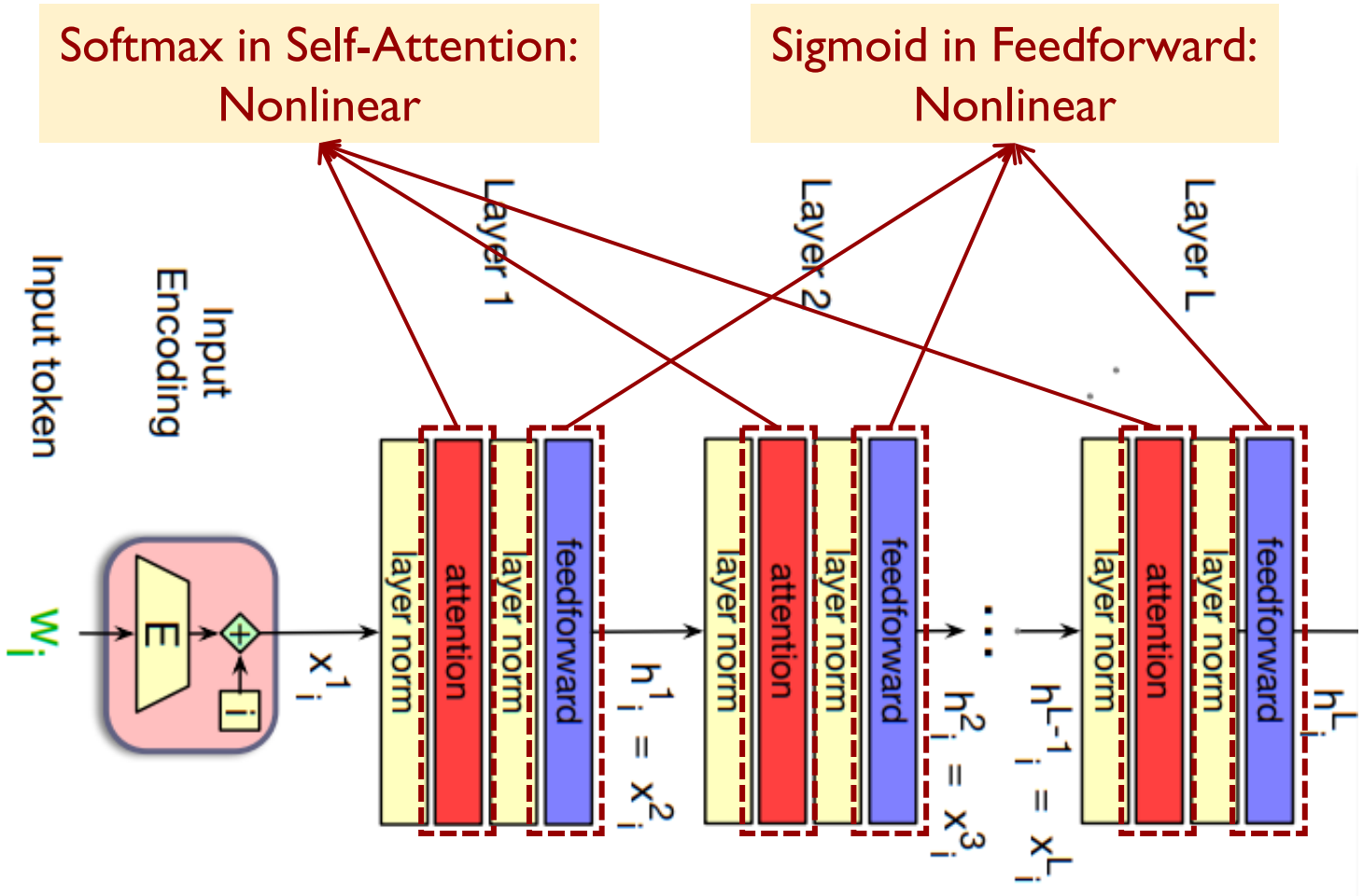
- [Levy and Golberg, NIPS 2014] word2vec is mathematically equivalent to factorizing a word-word matrix U, where

Nonlinear

$$U_{xy} = \log \frac{\#(x,y) \cdot |\mathcal{D}|}{\#(x) \cdot \#(y) \cdot b}$$

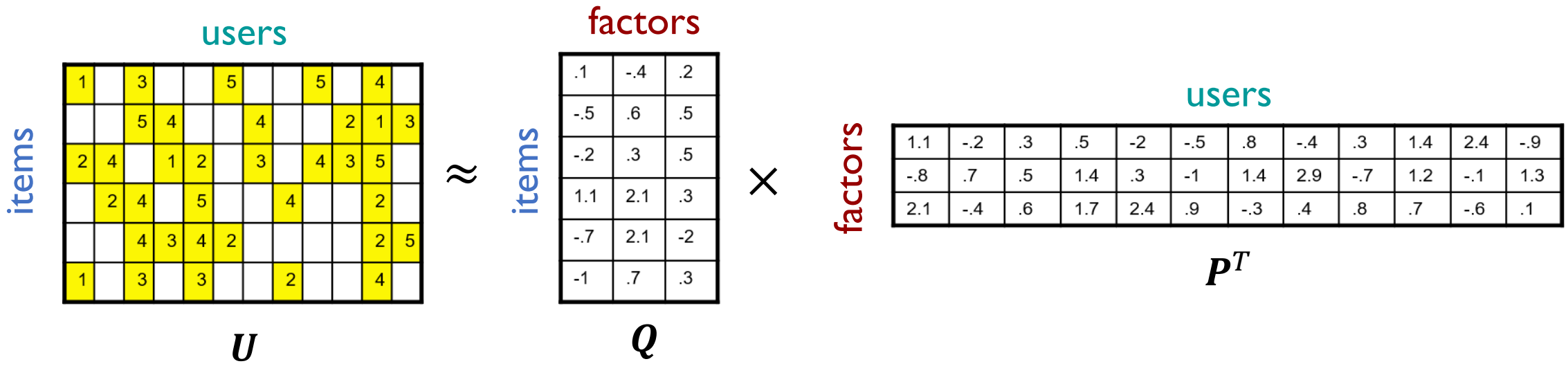
Neural Language Models (BERT)

- Transformer is a neural network



Nonlinearity in Recommender Systems?

- Do the recommender system techniques we have learned so far contain enough nonlinearity?
  - No, matrix factorization is essentially modeling the inner product of vectors, which is a highly linear operation.



$$U_{xi} = \boldsymbol{q}_i \boldsymbol{p}_x^T$$

Neural Collaborative Filtering [He et al., WWW 2017]

Neural Collaborative Filtering*

Xiangnan He
National University of Singapore, Singapore
xiangnanhe@gmail.com

Lizi Liao
National University of Singapore, Singapore
liaolizi.llz@gmail.com

Hanwang Zhang
Columbia University
USA
hanwangzhang@gmail.com

Liqiang Nie
Shandong University
China
nieliqiang@gmail.com

Xia Hu
Texas A&M University
USA
hu@cse.tamu.edu

Tat-Seng Chua
National University of Singapore, Singapore
dcscts@nus.edu.sg

ABSTRACT

In recent years, deep neural networks have yielded immense success on speech recognition, computer vision and natural language processing. However, the exploration of deep neural networks on recommender systems has received relatively less scrutiny. In this work, we strive to develop techniques based on neural networks to tackle the key problem in recommendation — collaborative filtering — on the basis of implicit feedback.

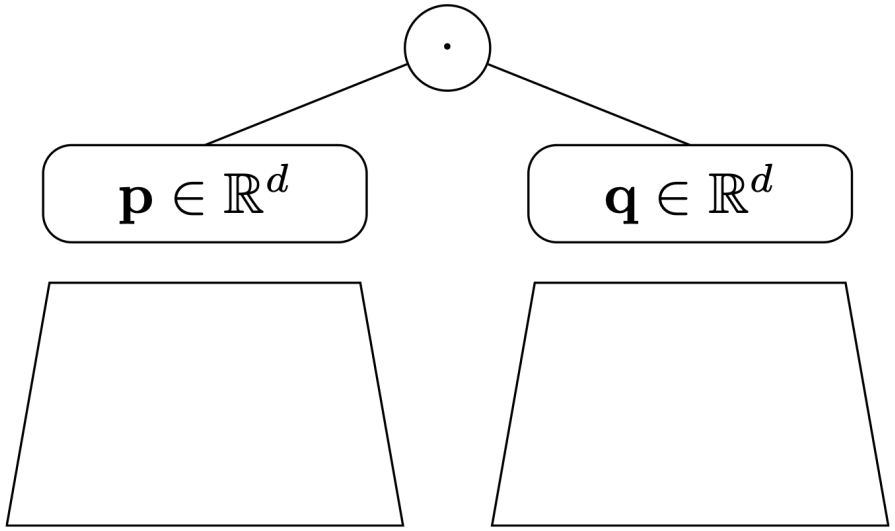
1. INTRODUCTION

In the era of information explosion, recommender systems play a pivotal role in alleviating information overload, having been widely adopted by many online services, including E-commerce, online news and social media sites. The key to a personalized recommender system is in modelling users' preference on items based on their past interactions (*e.g.*, ratings and clicks), known as collaborative filtering [31, 46]. Among the various collaborative filtering techniques, matrix

Learning the Similarity Function

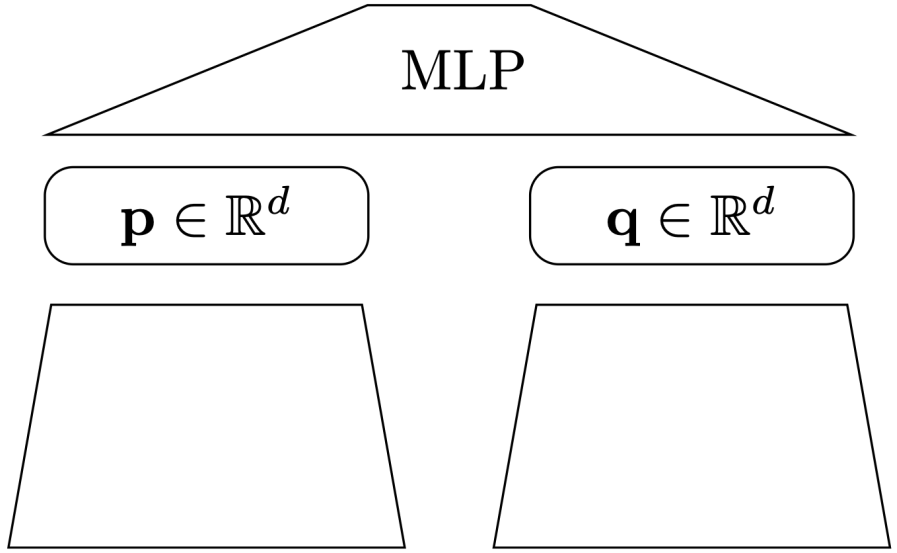
- Predefined similarity: inner product

$$\phi^{\text{dot}}(\mathbf{p}, \mathbf{q}) = \langle \mathbf{p}, \mathbf{q} \rangle$$



- Learned similarity: multi-layer perceptron (MLP)

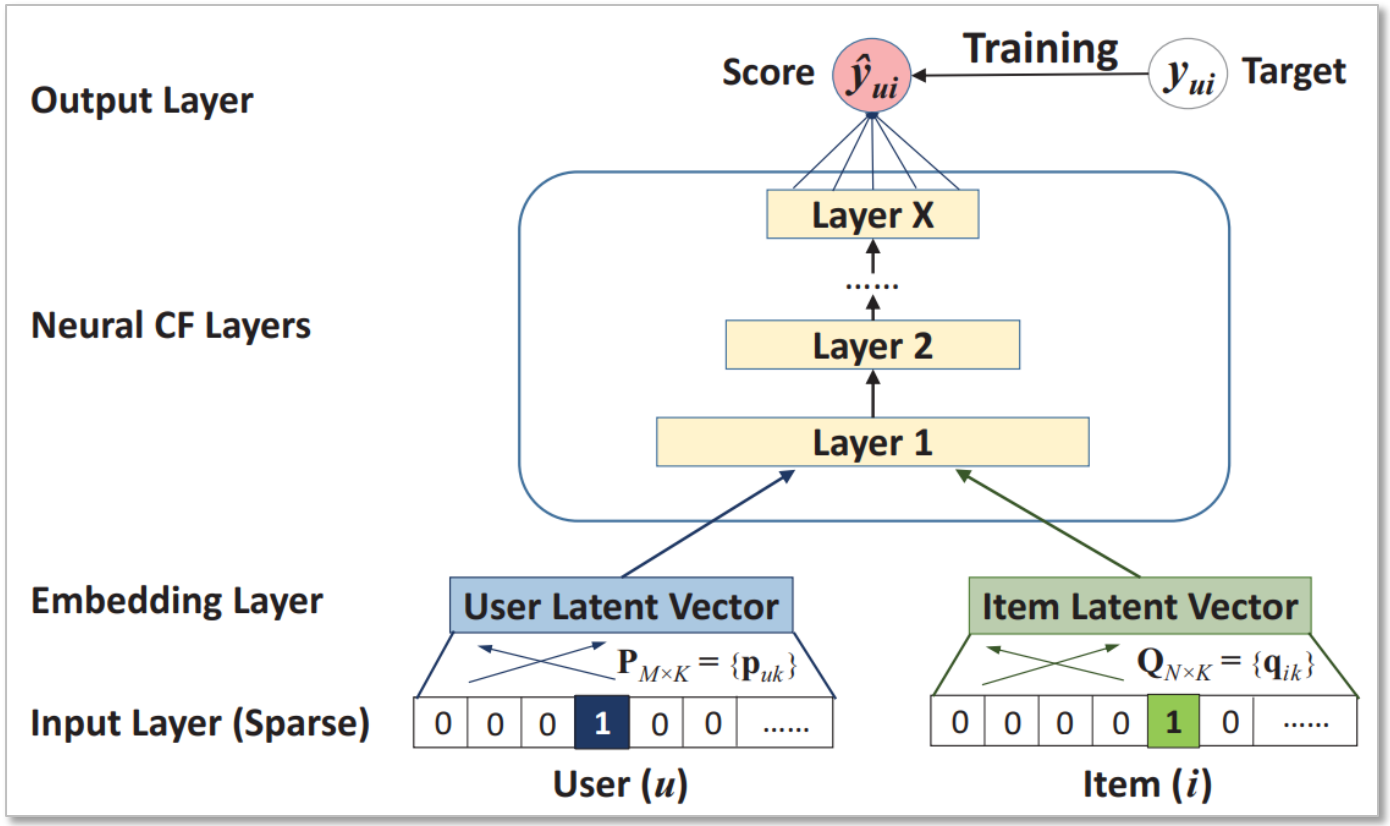
$$\phi^{\text{MLP}}(\mathbf{p}, \mathbf{q}) = \mathbf{f}_{W_l, \mathbf{b}_l}(\ldots \mathbf{f}_{W_1, \mathbf{b}_1}([\mathbf{p}, \mathbf{q}]) \ldots)$$



- We could use any sort of embeddings down here (e.g., from SVD or Matrix Factorization)

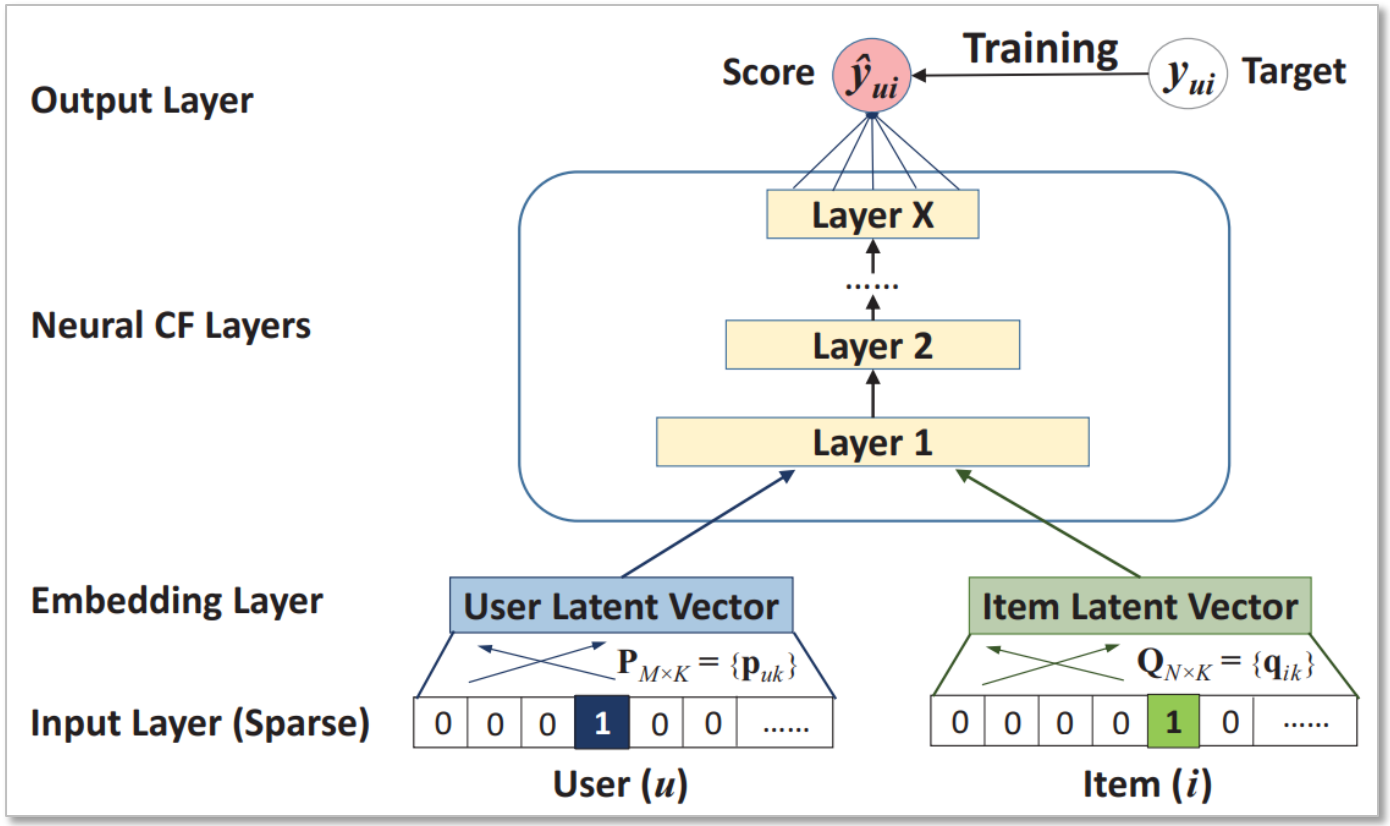
Example

- Suppose after matrix factorization,
  - User u's vector: $\boldsymbol{p} = [2, 2]$
  - Item i's vector: $\boldsymbol{q} = [2, 3]$
- Inner product: $\text{score}(u, i) = \boldsymbol{p}\boldsymbol{q}^T = 10$
- MLP (assuming 3 layers):
  - Input $\boldsymbol{x}_0 = [\boldsymbol{p}, \boldsymbol{q}] = [2, 2, 2, 3]$
  - $\boldsymbol{x}_1 = \text{Sigmoid}(\boldsymbol{W}_1\boldsymbol{x}_0 + \boldsymbol{b}_1)$
  - $\boldsymbol{x}_2 = \text{Sigmoid}(\boldsymbol{W}_2\boldsymbol{x}_1 + \boldsymbol{b}_2)$
  - $\hat{y} = \text{score}(u, i) = \text{Sigmoid}(\boldsymbol{W}_3\boldsymbol{x}_2)$



Example

- MLP (assuming 3 layers):
  - Input $\boldsymbol{x}_0 = [\boldsymbol{p}, \boldsymbol{q}] = [2, 2, 2, 3]$
  - $\boldsymbol{x}_1 = \text{Sigmoid}(\boldsymbol{W}_1 \boldsymbol{x}_0 + \boldsymbol{b}_1)$
  - $\boldsymbol{x}_2 = \text{Sigmoid}(\boldsymbol{W}_2 \boldsymbol{x}_1 + \boldsymbol{b}_2)$
  - $\hat{y} = \text{Sigmoid}(\boldsymbol{W}_3 \boldsymbol{x}_2)$
- Parameters?
  - $\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{b}_2, \boldsymbol{W}_3$
  - $\boldsymbol{P}, \boldsymbol{Q}$
- Learning objective?
  - RMSE $= (y - \hat{y})^2$
- Training data?
  - Known ratings in the user-item matrix



Combining Matrix Factorization and MLP

- Inner product: $\text{score}(u, i) = \boldsymbol{p}\boldsymbol{q}^T = 10$

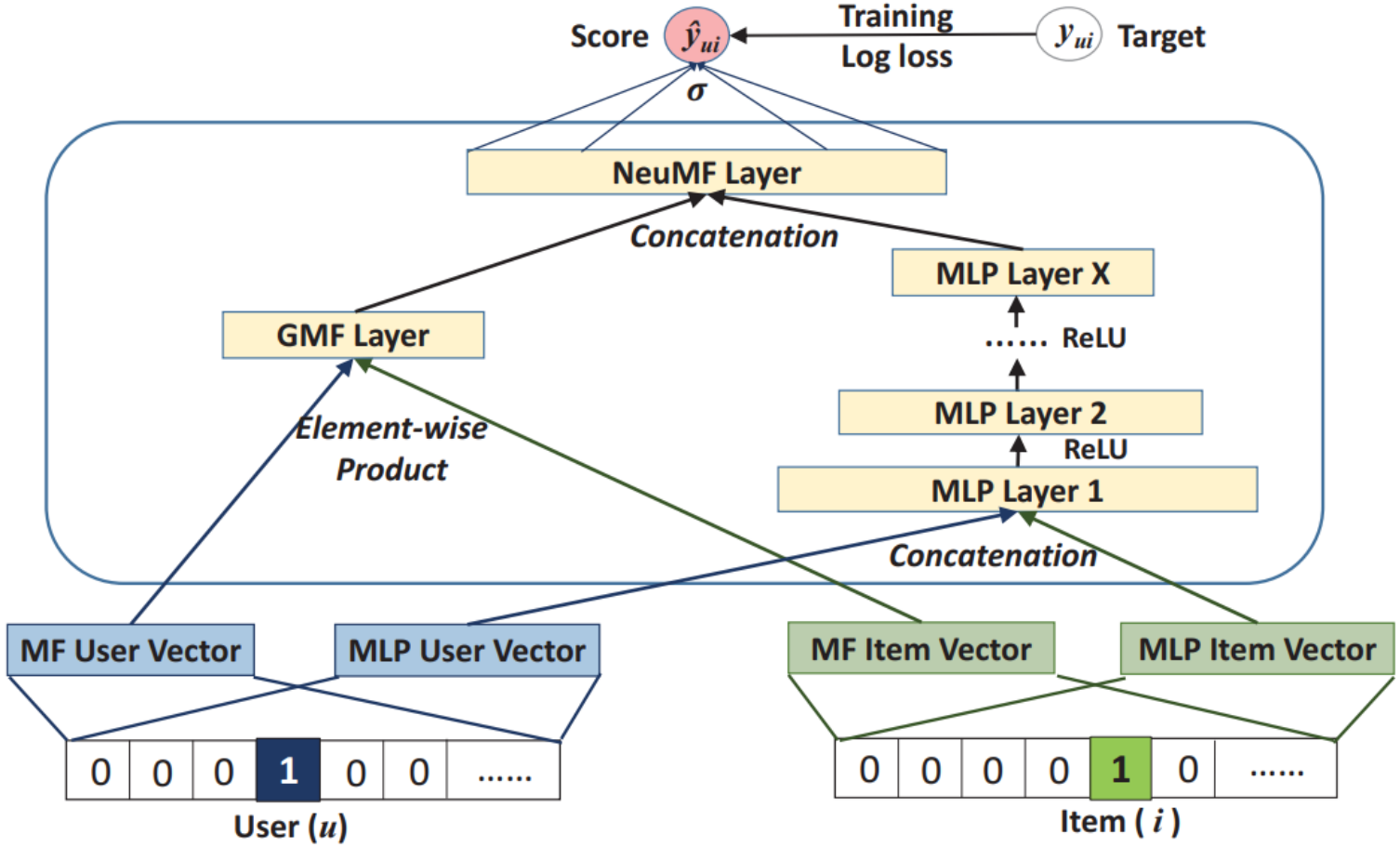
- MLP (assuming 3 layers):
  - Input $\boldsymbol{x}_0 = [\boldsymbol{p}, \boldsymbol{q}] = [2, 2, 2, 3]$
  - $\boldsymbol{x}_1 = \text{Sigmoid}(\boldsymbol{W}_1\boldsymbol{x}_0 + \boldsymbol{b}_1)$
  - $\boldsymbol{x}_2 = \text{Sigmoid}(\boldsymbol{W}_2\boldsymbol{x}_1 + \boldsymbol{b}_2)$
  - $\text{score}(u, i) = \text{Sigmoid}(\boldsymbol{W}_3\boldsymbol{x}_2)$

- MLP does not explicitly model the interaction between $\boldsymbol{p}$ and $\boldsymbol{q}$, but inner product does!
- Can we fuse inner product and MLP?

$$\text{score}(u, i) = \text{score}_{\text{MF}}(u, i) + \text{score}_{\text{MLP}}(u, i)$$

Combining Matrix Factorization and MLP

- User u's vector: $\boldsymbol{p} = [2, 2]$
- Item i's vector: $\boldsymbol{q} = [2, 3]$
- MLP: same
- Generalized Matrix Factorization (GMF):
  - Step 1: Element-wise product $\boldsymbol{p} \odot \boldsymbol{q} = [4, 6]$
  - Step 2: Nonlinear layer $\text{score}_{\text{MF}}(u, i) = \text{Sigmoid}(\boldsymbol{w} \begin{bmatrix} 4 \\ 6 \end{bmatrix})$
  - $\boldsymbol{w}$ are the parameters to be learned



Generalized Matrix Factorization vs. Inner Product

- User u's vector: $\boldsymbol{p} = [2, 2]$
- Item i's vector: $\boldsymbol{q} = [2, 3]$
- Generalized Matrix Factorization (GMF):
  - Step 1: Element-wise product $\boldsymbol{p} \odot \boldsymbol{q} = [4, 6]$
  - Step 2: Nonlinear layer $\text{score}_{\text{MF}}(u, i) = \text{Sigmoid}(\boldsymbol{w} \begin{bmatrix} 4 \\ 6 \end{bmatrix})$
- Suppose $\boldsymbol{w} = [1, -1]$, then

$$\boldsymbol{w}(\boldsymbol{p} \odot \boldsymbol{q}) = [1, -1] \begin{bmatrix} 2 \times 2 \\ 2 \times 3 \end{bmatrix} = 1 \times 2 \times 2 + (-1) \times 2 \times 3$$

$$= [2, 2] \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \boldsymbol{p}\ \text{diag}\{\boldsymbol{w}\}\ \boldsymbol{q}^T$$

$$\boxed{\text{score}_{\text{MF}}(u, i) = \text{Sigmoid}(\boldsymbol{p}\ \text{diag}\{\boldsymbol{w}\}\ \boldsymbol{q}^T)}$$

Nonlinear

Learnable Parameters

Summary

Generalized Matrix Factorization:

Input $\boldsymbol{x}_0 = [\boldsymbol{p}, \boldsymbol{q}]$

$$\text{score}_{\text{MF}}(u, i) = \text{Sigmoid}\big(\boldsymbol{w}(\boldsymbol{p} \odot \boldsymbol{q})\big) = \text{Sigmoid}(\boldsymbol{p}\ \text{diag}\{\boldsymbol{w}\}\ \boldsymbol{q}^T)$$

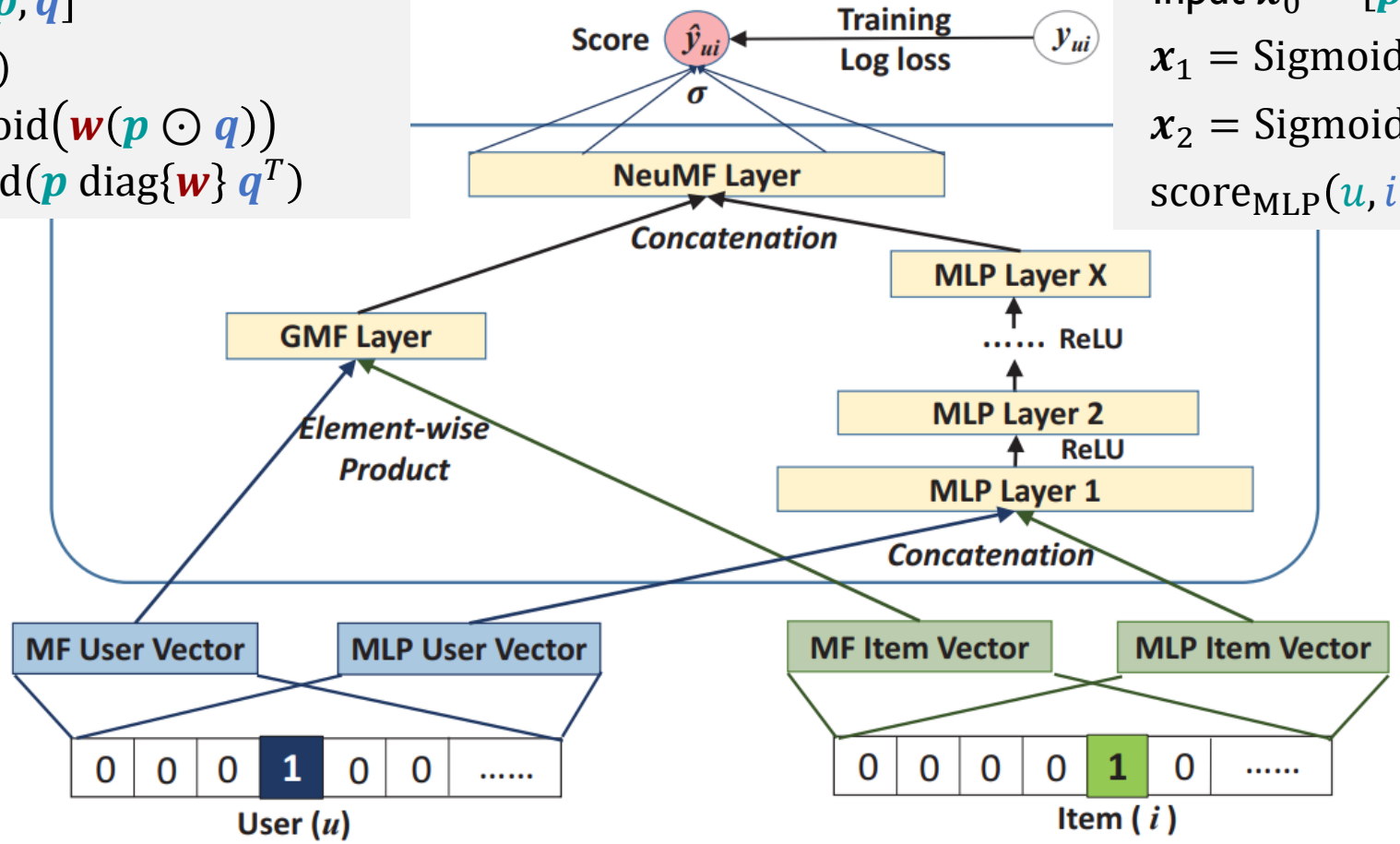
MLP (assuming 3 layers):

Input $\boldsymbol{x}_0 = [\boldsymbol{p}, \boldsymbol{q}]$

$$\boldsymbol{x}_1 = \text{Sigmoid}(\boldsymbol{W}_1 \boldsymbol{x}_0 + \boldsymbol{b}_1)$$

$$\boldsymbol{x}_2 = \text{Sigmoid}(\boldsymbol{W}_2 \boldsymbol{x}_1 + \boldsymbol{b}_2)$$

$$\text{score}_{\text{MLP}}(u, i) = \text{Sigmoid}(\boldsymbol{W}_3 \boldsymbol{x}_2)$$

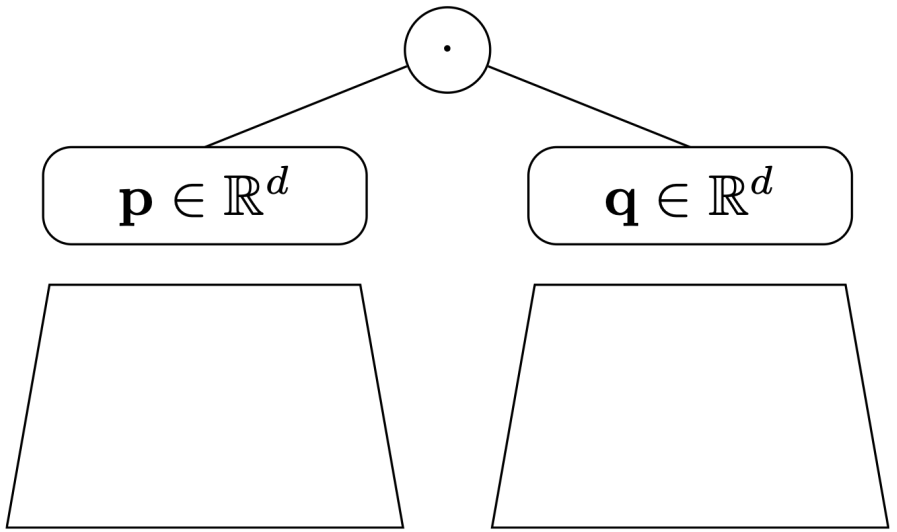


$$\text{score}(u, i) = \text{score}_{\text{MF}}(u, i) + \text{score}_{\text{MLP}}(u, i)$$

Matrix Factorization vs. Neural Collaborative Filtering

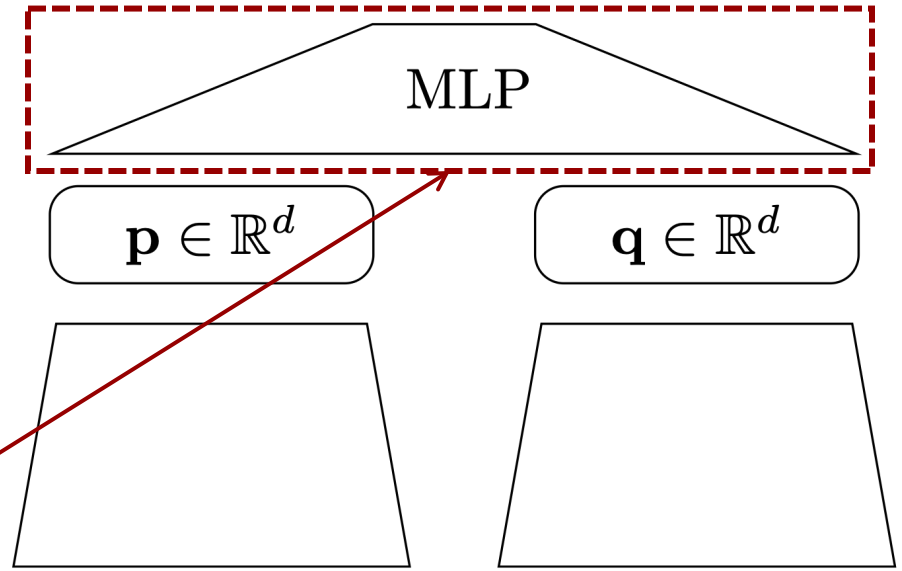
- Predefined similarity

$$\phi^{\text{dot}}(\mathbf{p}, \mathbf{q}) = \langle \mathbf{p}, \mathbf{q} \rangle$$



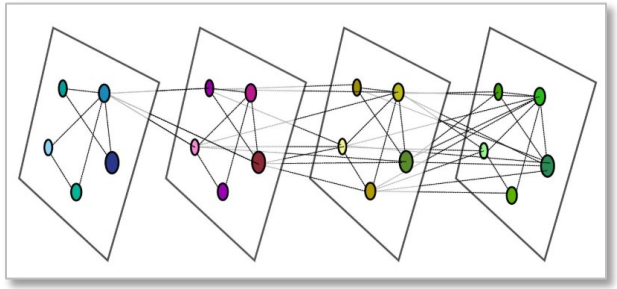
- Learned similarity

$$\phi^{\text{MLP}}(\mathbf{p}, \mathbf{q}) = \mathbf{f}_{W_l, \mathbf{b}_l}(\ldots \mathbf{f}_{W_1, \mathbf{b}_1}([\mathbf{p}, \mathbf{q}]) \ldots)$$



We can make this part even more complicated!

(E.g., graph neural networks)



But is inner product really that bad?

Performance Revisited [Rendle et al., RecSys 2020]

Neural Collaborative Filtering vs. Matrix Factorization Revisited

Steffen Rendle
srendle@google.com
Google Research
Mountain View, California

Walid Krichene
walidk@google.com
Google Research
Mountain View, California

Li Zhang
liqzhang@google.com
Google Research
Mountain View, California

John Anderson
janders@google.com
Google Research
Mountain View, California

ABSTRACT

Embedding based models have been the state of the art in collaborative filtering for over a decade. Traditionally, the dot product or higher order equivalents have been used to combine two or more embeddings, e.g., most notably in matrix factorization. In recent years, it was suggested to replace the dot product with a learned similarity e.g. using a multilayer perceptron (MLP). This approach is often referred to as *neural collaborative filtering* (NCF). In this work, we revisit the experiments of the NCF paper that popularized learned similarities using MLPs. First, we show that with a

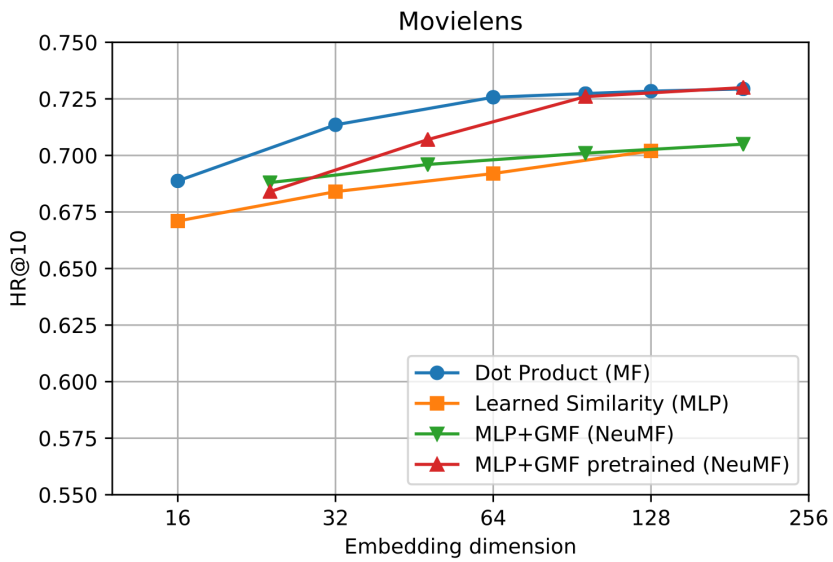
1 INTRODUCTION

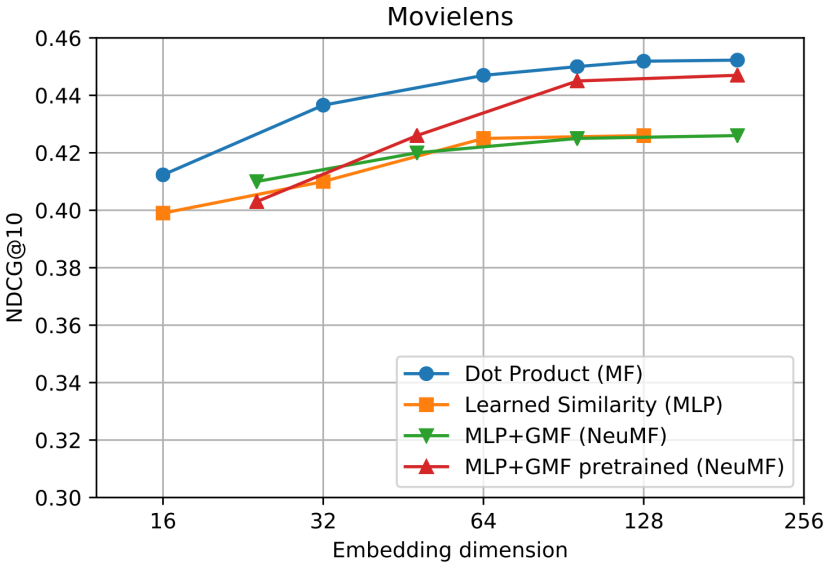
Embedding based models have been the state of the art in collaborative filtering for over a decade. A core operation of most of these embedding based models is to combine two or more embeddings. For example, combining a user embedding with an item embedding to obtain a single score that indicates the preference of the user for the item. This can be viewed as a *similarity function* in the embedding space. Traditionally, a dot product or higher order products have been used for the similarity. Recently, it has become popular to learn the similarity function with a neural network. Most

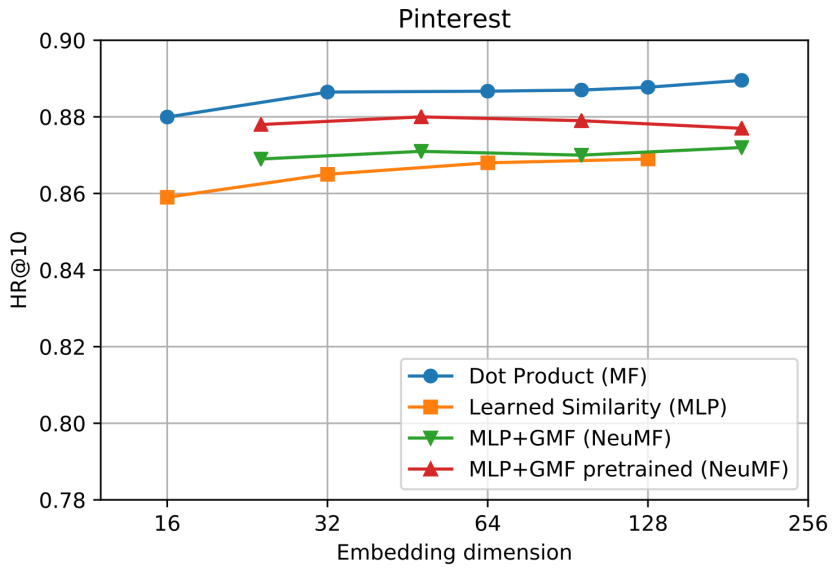
Performance Revisited

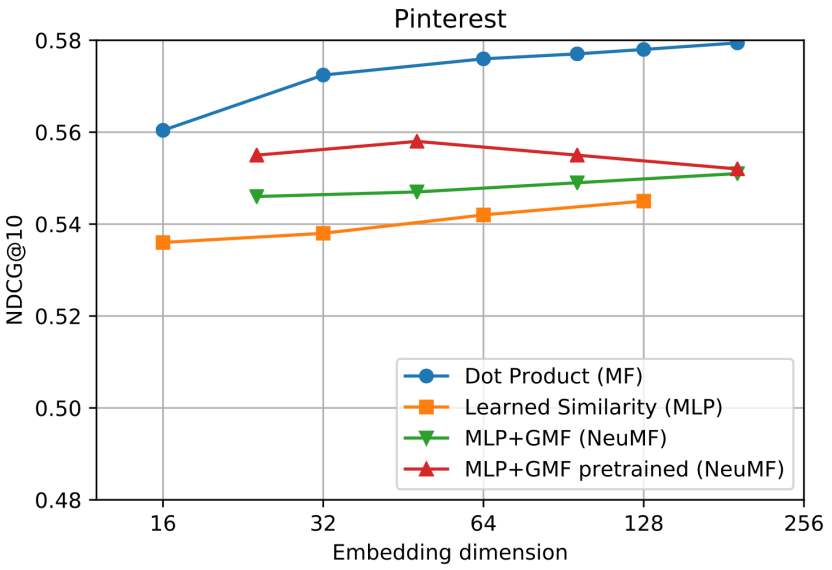
- Motivation: Learning similarity functions (e.g., Neural Collaborative Filtering) has typically been regarded as the de facto standard in academic studies and is considered a strong baseline
- Experimental Setup
  - Follows the Neural Collaborative Filtering paper
  - Datasets: MovieLens 1M and Pinterest
  - Compared Methods: Matrix Factorization (MF) and Neural Collaborative Filtering variants

Results







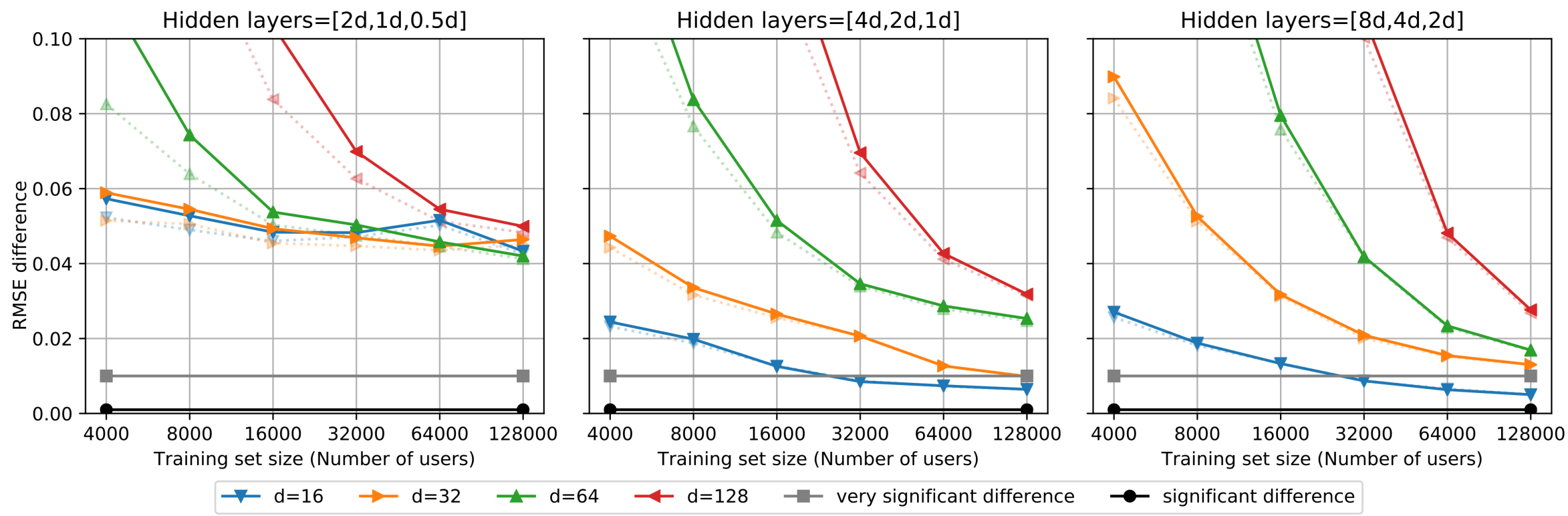


- MF substantially outperforms Neural Collaborative Filtering variants (MLP and NeuMF) on all datasets, evaluation metrics and embedding dimensions.

Why does this happen?

- Why did we believe Neural Collaborative Filtering could be better than MF?
  - Since MLPs can approximate any continuous function under certain conditions (known as the Universal Approximation Theorem), we expect MLPs to capture more complex user-item similarity patterns than a simple inner product
- But how much training data does Neural Collaborative Filtering need to capture complex user-item similarity patterns?
- Let's further simplify the learning problem: how much training data does Neural Collaborative Filtering need to capture even the simple inner product similarity?

MLPs require massive data even to learn an inner product!

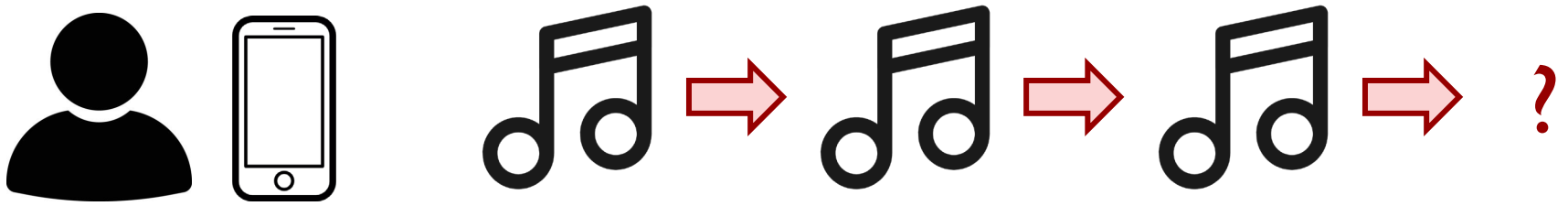


- As models become more complex, we need more training data (for example, the entire Wikipedia and BookCorpus used for BERT). However, in recommender systems, we rarely have access to such large amounts of data.

Sequential Recommendation

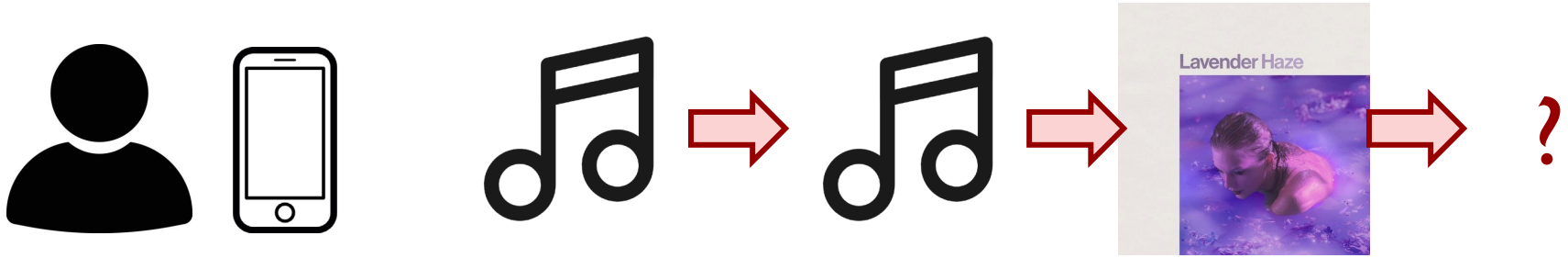
We are adding streaming to our record store!

- What is the next track to stream to a user?



We are adding streaming to our record store!

- What is the next track to stream to a user?



- Our user just listened to Taylor Swift's *Lavender Haze*
  - What should we do next?
  - Play more songs by Taylor Swift?
  - Play more color-themed songs?
  - Play more modern pop songs?
  - Play it again?

- Question 1: What do these options have in common?
- Question 2: What other factors should I consider in choosing among these options?

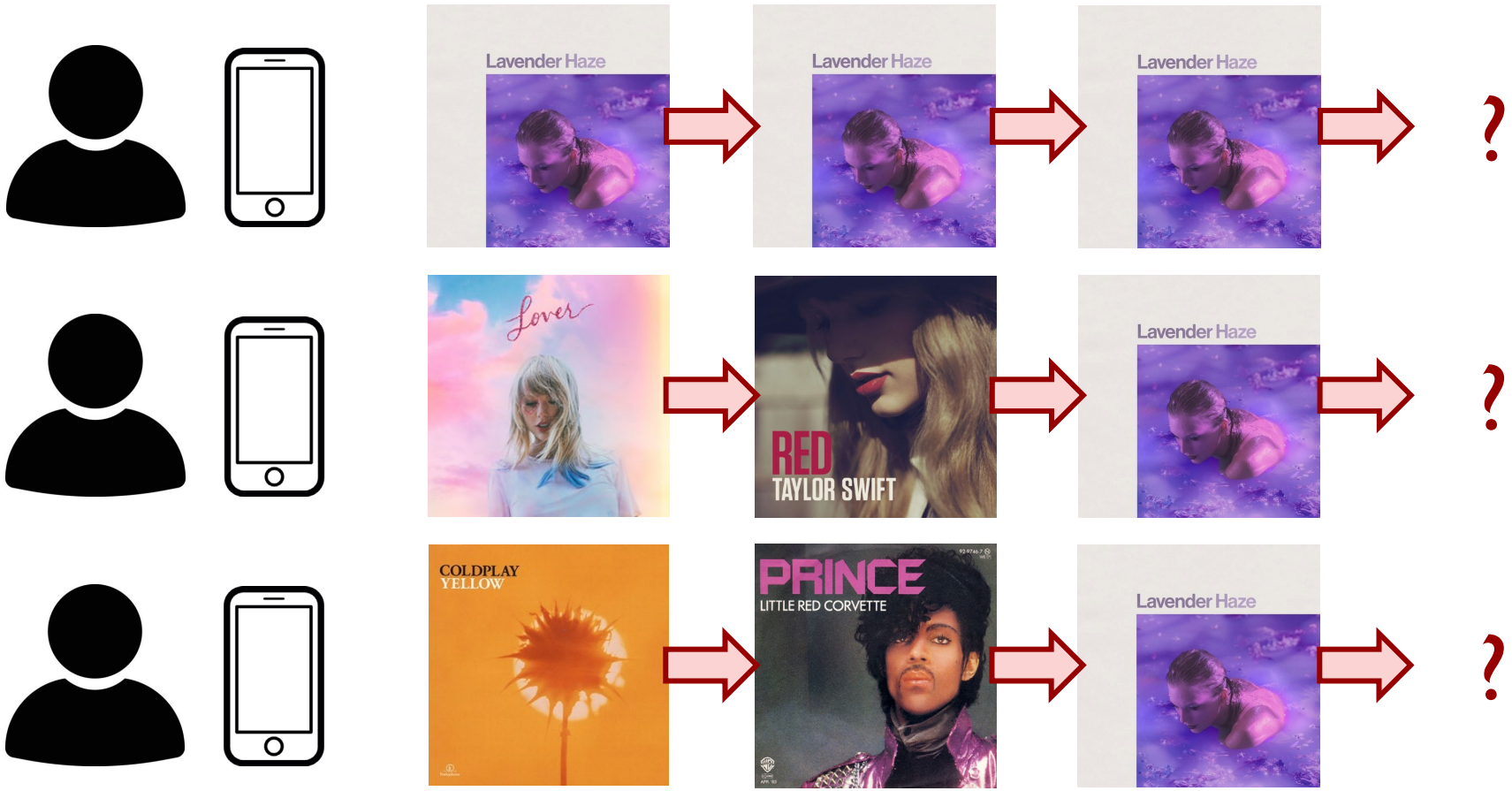
What RecSys techniques have we already learned?

- Content-Based Approach: Recommend items that are similar to those previously viewed
  - Here, the definition of similarity is based on attributes (e.g., lyrics)
- Item-Item Collaborative Filtering: Given a candidate item, first identify its similar items, and then check whether the user has viewed any of them
  - Here, the definition of similarity is based on the columns corresponding to the two items in the user-item matrix
- Matrix Factorization: Find the item in the embedding space that are closest to the user and have not yet been viewed
  - A user's embedding should be close to the embeddings of the items they have already viewed
- Neural Collaborative Filtering: More sophisticated similarity computation methods, but still based on embeddings

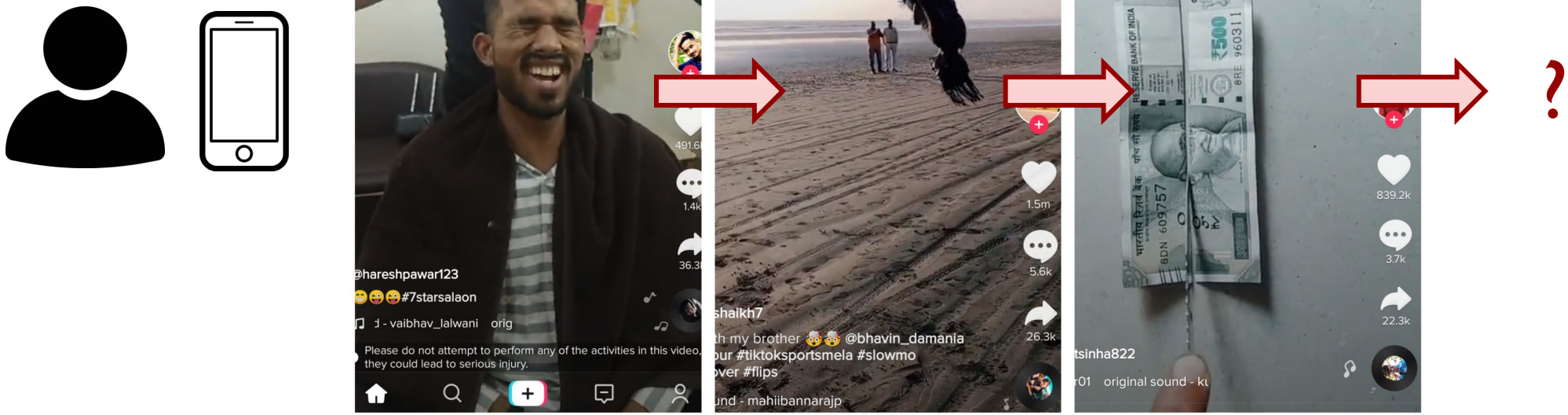
- Summary: Find items similar to those a user has already viewed. The only difference lies in how similarity is defined

What are we missing?

- Understanding dynamic user intent from sequential interactions



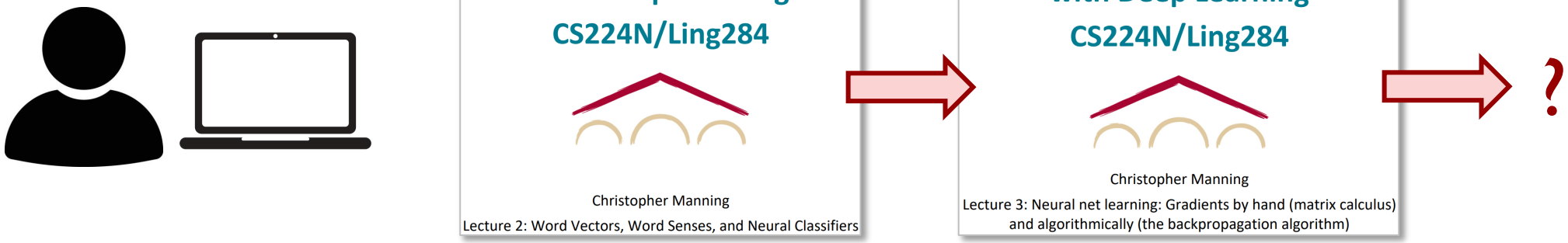
Other Sequential RecSys Examples: TikTok



Other Sequential RecSys Examples: Netflix

- For certain products, the order can be crucial!
  - [movies and books] Suppose I just watched *Harry Potter 2*. Would it make sense to recommend *Harry Potter 1*?
  - [lecture videos] Suppose I just watched the video for Lecture 3 of Stanford's CS224N. Would it make sense to recommend the video for Lecture 2?
  - [electronic devices] Suppose I just bought an iPhone XR. Would it make sense to recommend an iPhone 8?

Other Sequential RecSys Examples: YouTube and Amazon



What are we missing?

- Embedding similarity is symmetric.
  - If the user has just viewed item i, whether the model will recommend item j as the next item fully depends on $\text{score}(i, j) = \boldsymbol{e}_i^T \boldsymbol{e}_j$ or $\cos(\boldsymbol{e}_i, \boldsymbol{e}_j)$
  - If the user has just viewed item j, whether the model will recommend item i as the next item fully depends on $\text{score}(j, i) = \text{score}(i, j)$
- Sequential recommendation is asymmetric.
  - When your model recommends item j as the next item after item i, it does NOT necessarily mean that i should be recommended as the next item after j.
  - We should model something like $\text{score}(j \mid i) \neq \text{score}(i \mid j)$

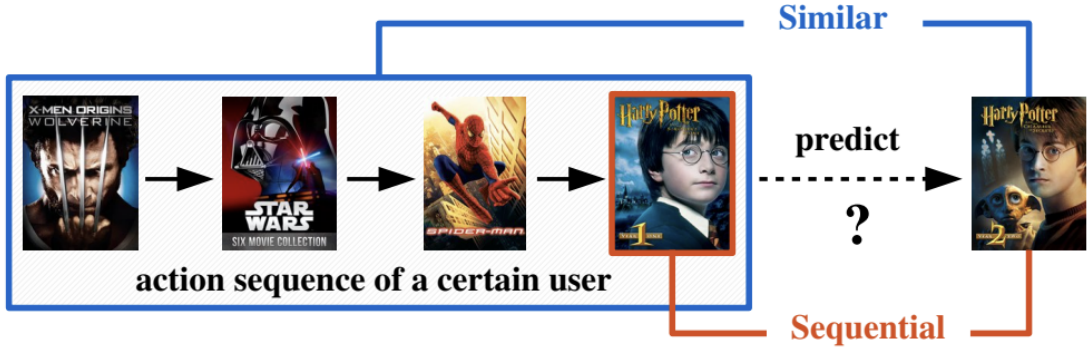
Transition matrix is asymmetric!

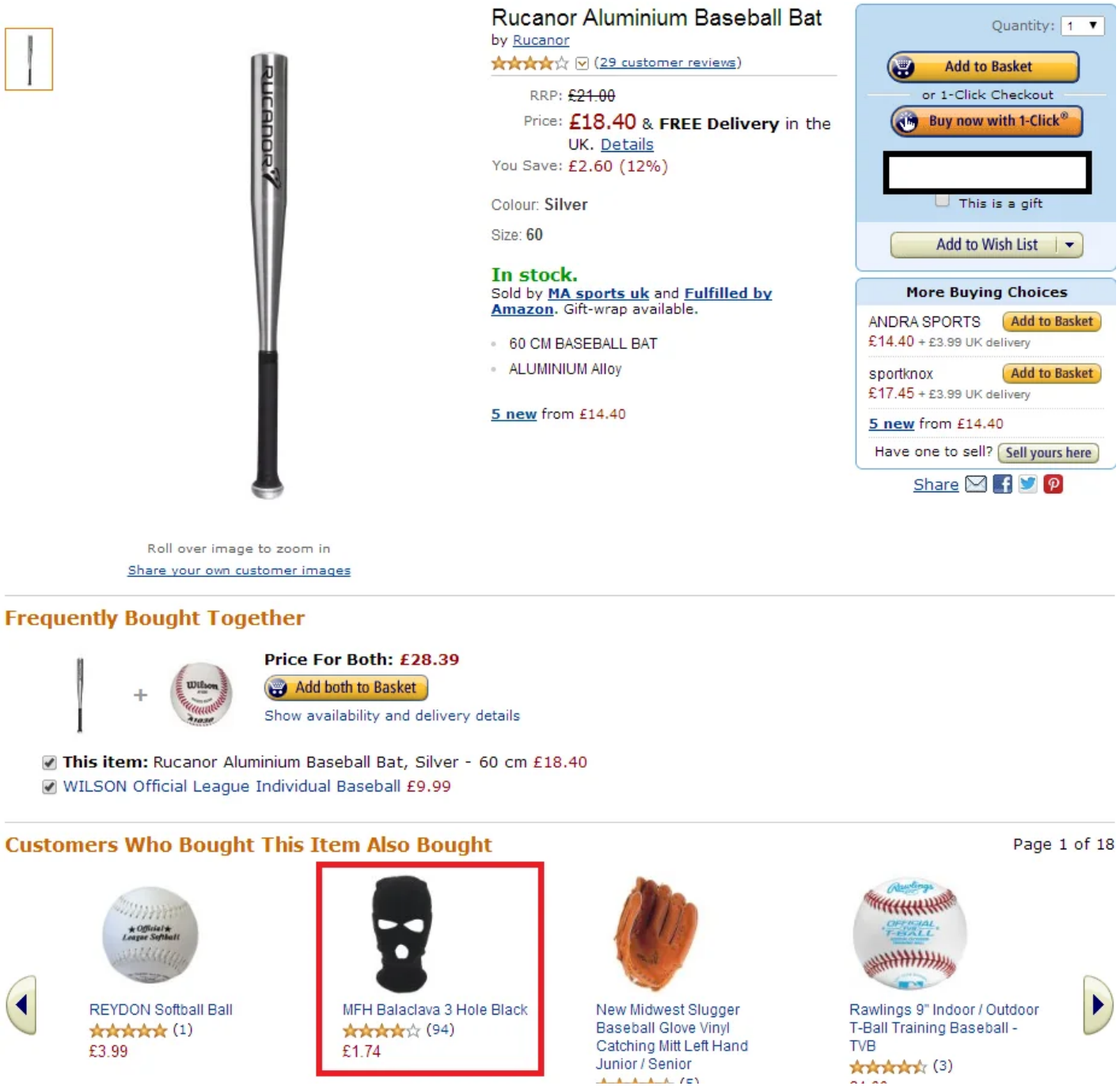
Fossil [He and McAuley, ICDM 2016]

Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation

Ruining He, Julian McAuley
Department of Computer Science and Engineering
University of California, San Diego
Email: {r4he, jmcauley}@cs.ucsd.edu

***Abstract*—Predicting personalized sequential behavior is a key task for recommender systems. In order to predict user actions such as the next product to purchase, movie to watch, or place to visit, it is essential to take into account both long-term user preferences and sequential patterns (i.e., short-term dynamics). Matrix Factorization and Markov Chain methods have emerged as two separate but powerful paradigms for modeling the two respectively. Combining these ideas has led**





- Predict a user's next step based on the next-step behavior patterns of the overall population
- Amazon's "*Customer Who Bought This Item Also Bought*"

Markov Chain to Model Sequential Dynamics

- Historical user-item interactions

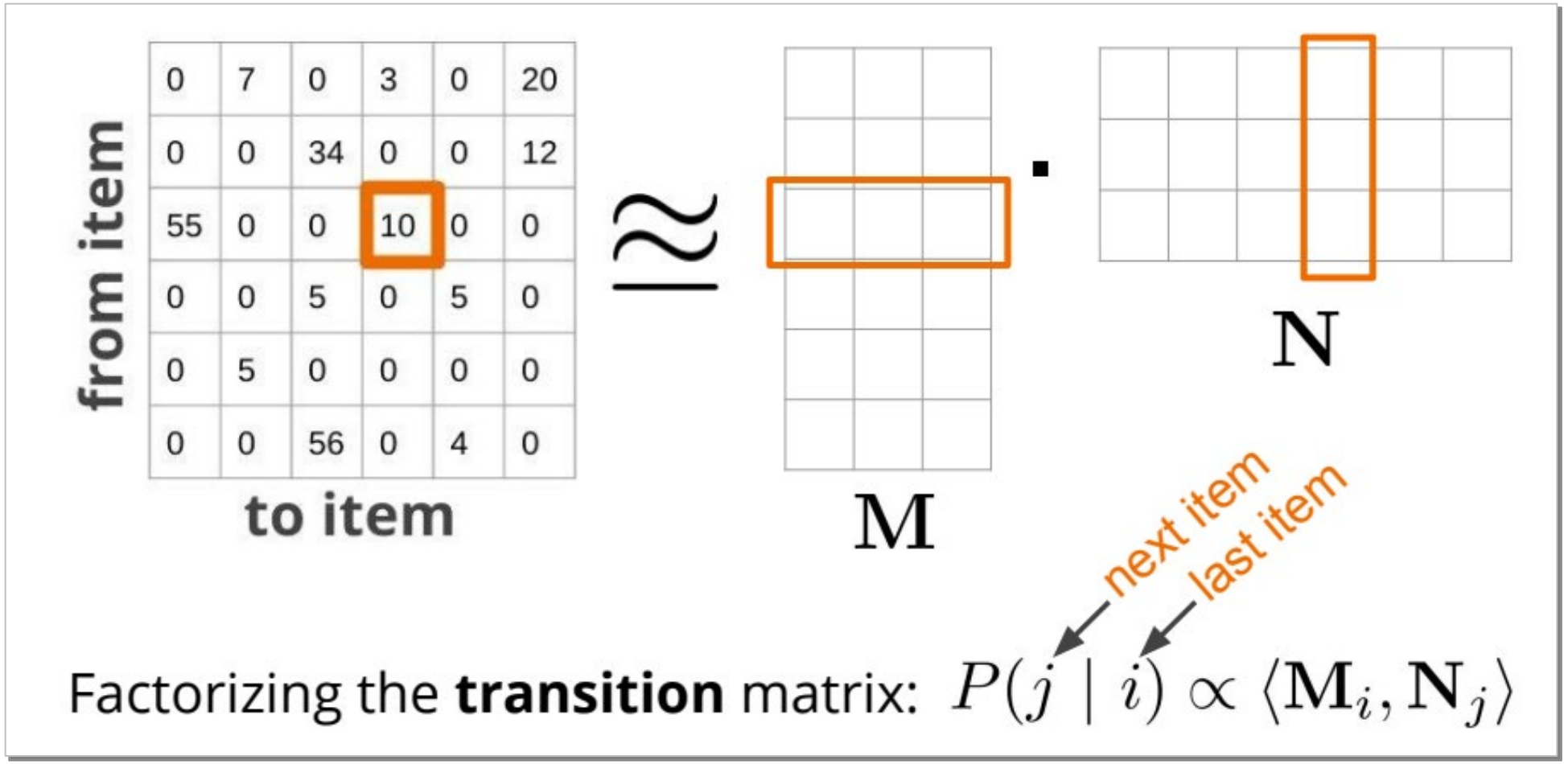
User								
$u1$	$\rightarrow$	$i1$	$\rightarrow$	$i2$	$\rightarrow$	$i4$		
$u2$	$\rightarrow$	$i3$	$\rightarrow$	$i4$				
$u3$	$\rightarrow$	$i1$	$\rightarrow$	$i2$	$\rightarrow$	$i4$	$\rightarrow$	$i3$
$u4$	$\rightarrow$	$i3$	$\rightarrow$	$i1$	$\rightarrow$	$i2$		

- Step 1: Construct a transition matrix

	$i1$	$i2$	$i3$	$i4$
$i1$	0	3	0	0
$i2$	0	0	0	2
$i3$	1	0	0	1
$i4$	0	0	1	0

Markov Chain to Model Sequential Dynamics

- Step 2: Factorize the transition matrix



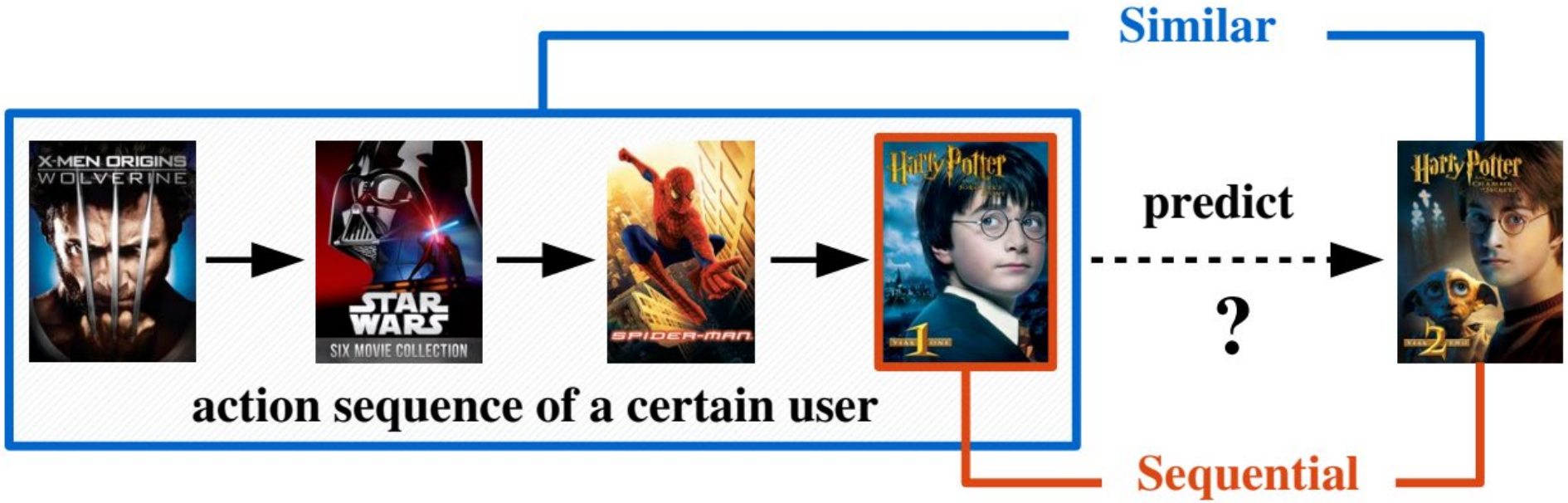
Markov Chain to Model Sequential Dynamics

- Problems with this approach?
  - NOT personalized!
  - Regardless of which user the model is considering, $\text{score}(j \mid i) \propto \boldsymbol{M}_i^T \boldsymbol{N}_j$
  - But $\boldsymbol{M}_i$ and $\boldsymbol{N}_j$ are the same for all users

- How to make the model personalized?

$$p_u(j \mid i) \propto \overbrace{\sum_{j' \in \mathcal{I}_u^+ \setminus \{j\}} \langle \mathbf{P}_{j'}, \mathbf{Q}_j \rangle}^{\text{user preferences}} + \underbrace{(\eta + \eta_u)}_{\substack{\text{personalized} \\ \text{weighting factor}}} \cdot \overbrace{\langle \mathbf{M}_i, \mathbf{N}_j \rangle}^{\text{sequential dynamics}}$$

$$p_u(j \mid i) \propto \overbrace{\sum_{j' \in \mathcal{I}_u^+ \setminus \{j\}} \langle \mathbf{P}_{j'}, \mathbf{Q}_j \rangle}^{\text{user preferences}} + \underbrace{(\eta + \eta_u)}_{\substack{\text{personalized} \\ \text{weighting factor}}} \cdot \overbrace{\langle \mathbf{M}_i, \mathbf{N}_j \rangle}^{\text{sequential dynamics}}$$



Back to Motivating Example

- Play *Lavender Haze* again? Play more songs by Taylor Swift? Play more color-themed songs?
  - Might all be likely according to the transition matrix. But if you consider user preference …

Higher-Order Markov Chain

User									
$u1$	$\rightarrow$	$i1$	$\rightarrow$	$i2$	$\rightarrow$	$i4$			
$u2$	$\rightarrow$	$i3$	$\rightarrow$	$i4$					
$u3$	$\rightarrow$	$i1$	$\rightarrow$	$i2$	$\rightarrow$	$i4$	$\rightarrow$	$i3$	
$u4$	$\rightarrow$	$i3$	$\rightarrow$	$i1$	$\rightarrow$	$i2$			

- Transition matrix:

	$i1$	$i2$	$i3$	$i4$
$i1 \rightarrow i2$	0	0	0	2
$i1 \rightarrow i3$	0	0	0	0
$i1 \rightarrow i4$	0	0	0	0
$i2 \rightarrow i1$	0	0	0	0
...	...	...	...	...

- Sparse
- # of rows increases exponentially with the order of the Markov chain
- Similar to the n-gram language model
- Can we use a neural language model to alleviate sparsity?

Performance

- Fossil outperforms BPR-MF (and other static baselines) because it models sequential dynamics
- Small orders seem to be enough to achieve good performance

Table IV: AUC on different datasets (higher is better). The number of latent dimensions for all comparison methods (except for POP) is set to $K = 10$. For *Fossil*, we test different orders of the Markov Chain (i.e., 1, 2, and 3). On the right we demonstrate the improvement of FPMC vs. BPR-MF, *Fossil* vs. FISM, *Fossil* vs. FPMC, and *Fossil* vs. the best baselines.

Dataset	(a) POP	(b) BPR-MF	(c) FISM	(d) FMC	(e) FPMC	(f-1) *Fossil*	(f-2) *Fossil*	(f-3) *Fossil*	% improvement			
									e vs. b	f vs. c	f vs. e	f vs. best
Amazon-Office	0.6427	0.6736	0.7113	0.6874	0.6891	0.7211	0.7224	0.7221	2.30%	1.56%	**4.83%**	1.56%
Amazon-Auto	0.5870	0.6379	0.6736	0.6452	0.6446	0.6910	0.6904	0.6901	1.05%	2.58%	**7.20%**	2.58%
Amazon-Game	0.7495	0.8483	0.8639	0.8401	0.8502	0.8793	0.8813	0.8817	0.22%	2.06%	**3.71%**	2.06%
Amazon-Toy	0.6240	0.7020	0.7499	0.6665	0.7061	0.7625	0.7645	0.7652	0.58%	2.04%	**8.37%**	2.04%
Amazon-Cell	0.6959	0.7212	0.7755	0.7359	0.7396	0.7982	0.8009	0.8006	2.55%	3.27%	**8.29%**	3.27%
Amazon-Clothes	0.6189	0.6513	0.7085	0.6673	0.6672	0.7255	0.7256	0.7259	2.44%	2.46%	**8.80%**	2.46%
Amazon-Elec	0.7837	0.7927	0.8210	0.7992	0.7985	0.8411	0.8438	0.8444	0.73%	2.85%	**5.75%**	2.85%
Epinions	0.4576	0.5520	0.5818	0.5532	0.5477	0.6014	0.6050	0.6048	-0.78%	3.99%	**10.46%**	3.99%
Foursquare	0.9168	0.9506	0.9230	0.9441	0.9485	0.9626	0.9621	0.9618	-0.22%	4.29%	**1.49%**	1.26%
Avg. ($K = 10$)	0.6751	0.7255	0.7565	0.7265	0.7324	0.7759	0.7773	0.7774	0.99%	2.79%	**6.54%**	2.45%
Avg. ($K = 20$)	0.6751	0.7285	0.7580	0.7293	0.7344	0.7780	0.7795	0.7788	0.88%	2.83%	**6.44%**	2.54%

Translation is asymmetric!

TransRec [He et al., RecSys 2017]

Translation-based Recommendation

Ruining He
UC San Diego
r4he@cs.ucsd.edu

Wang-Cheng Kang
UC San Diego
wckang@eng.ucsd.edu

Julian McAuley
UC San Diego
jmcauley@cs.ucsd.edu

ABSTRACT

Modeling the complex interactions between users and items as well as amongst items themselves is at the core of designing successful recommender systems. One classical setting is predicting users' personalized sequential behavior (or 'next-item' recommendation), where the challenges mainly lie in modeling 'third-order' interactions between a user, her previously visited item(s), and the next item to consume. Existing methods typically decompose these higher-order interactions into a combination of *pairwise* relationships, by way of which user preferences (user-item interactions) and sequential patterns (item-item interactions) are captured by separate components. In this paper, we propose a unified method, *TransRec*, to model such third-order relationships for large-scale sequen-

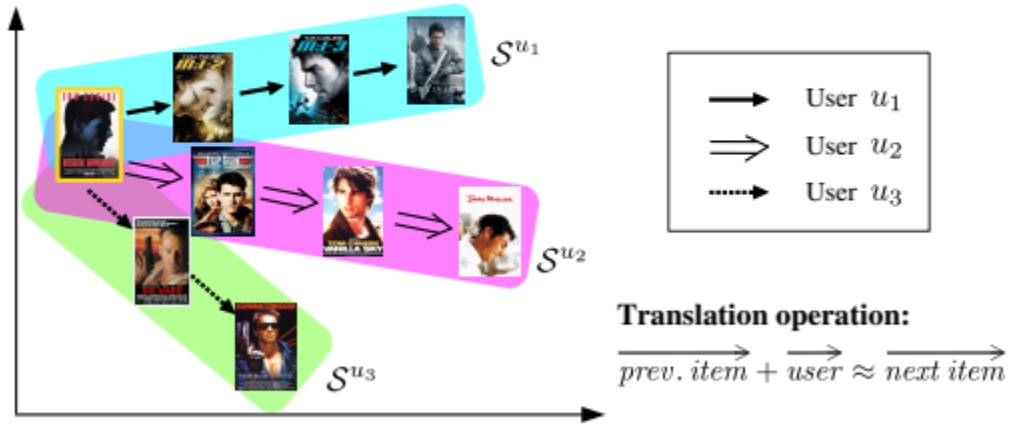
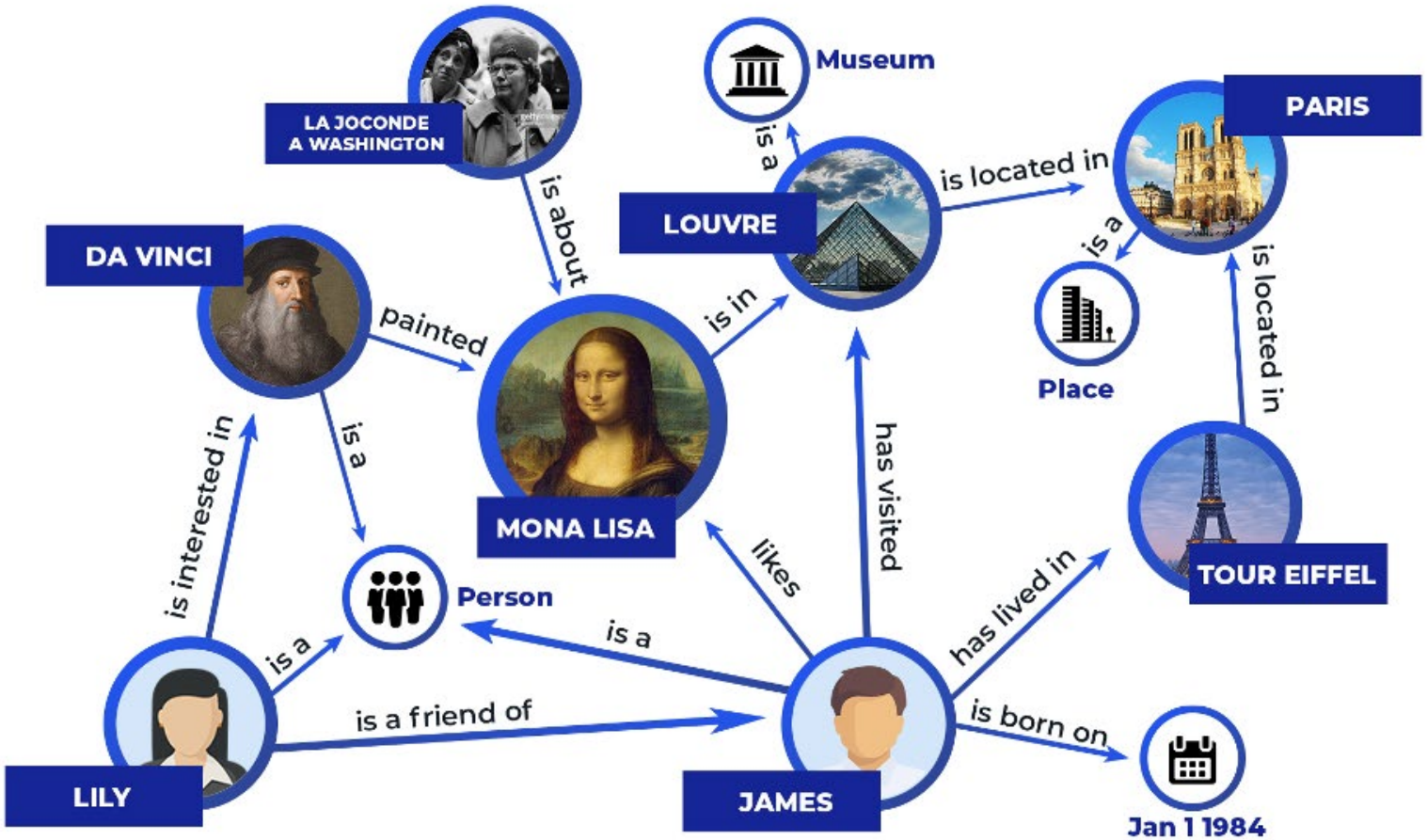
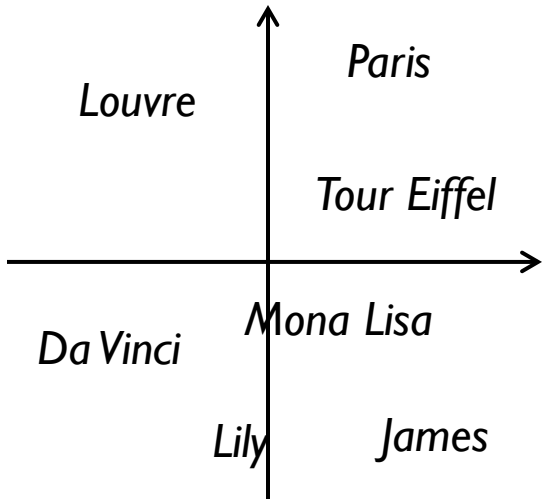


Figure 1: *TransRec* as a sequential model: Items (movies) are embedded into a 'transition space' where each user is modeled by a *translation* vector. The transition of a user from

Background: Translation-Based Embedding

- You have a knowledge graph, and you want to embed all the nodes into a vector space





Background: Translation-Based Embedding

- Each edge can be represented as a triplet (h, r, t)

head entity h	relation r	tail entity t
Tour Eiffel	is located in	*Paris*
Da Vinci	painted	*Mona Lisa*
Lily	is a friend of	*James*
…	…	…

- A simple (but suboptimal) solution: Each node should be close to its neighbors

$$\max_{\boldsymbol{e}_h, \boldsymbol{e}_t} \sum_{(h,r,t)\in\mathcal{G}} \frac{\exp(\boldsymbol{e}_h^T \boldsymbol{e}_t)}{\sum_{t'} \exp(\boldsymbol{e}_h^T \boldsymbol{e}_{t'})}$$

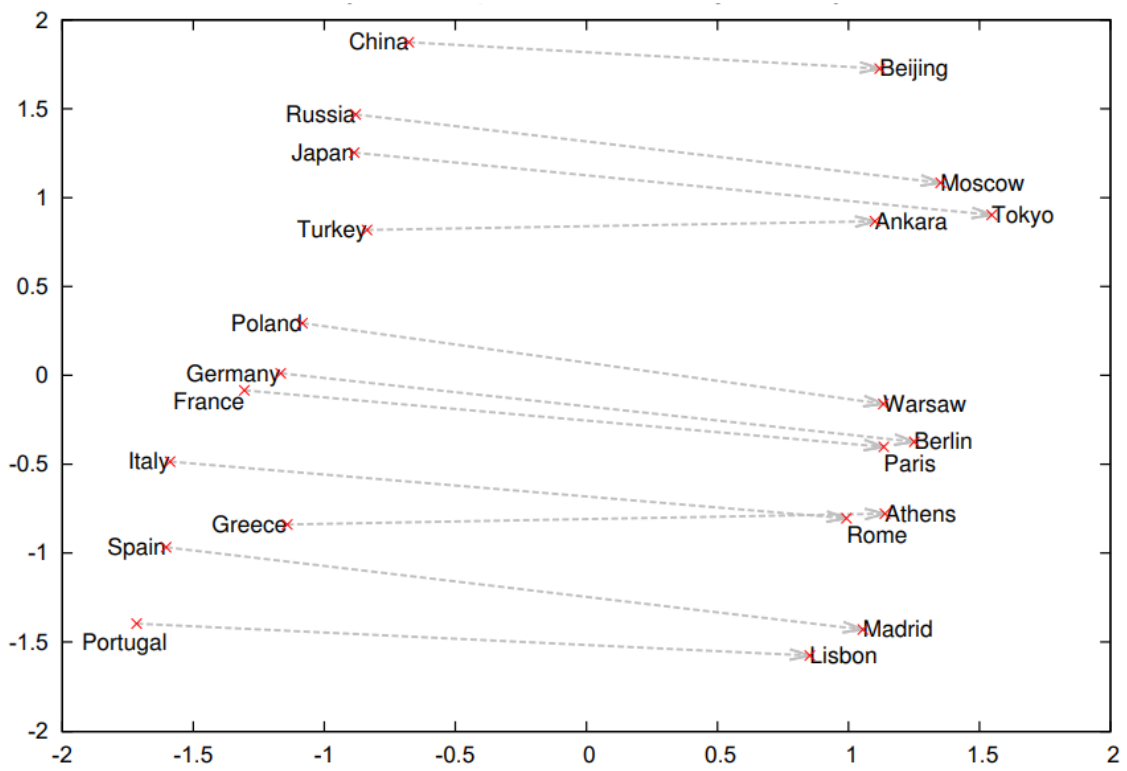
  - Limitation: It ignores different relations

TransE [Bordes et al., NIPS 2013]

- Idea: Model relations as translations

head entity h	relation r	tail entity t
Paris	is capital of	*France*

$$\boldsymbol{e}_h + \boldsymbol{e}_r \approx \boldsymbol{e}_t$$



TransE [Bordes et al., NIPS 2013]

- Idea: Model relations as translations
- Training:

head entity h	relation r	tail entity t
Paris	is capital of	*France*

Minimize $\|e_h + e_r - e_t\|$

head entity h	relation r	"corrupted" tail entity t'
Paris	is capital of	*UK*

Maximize $\|e_h + e_r - e_{t'}\|$

(but stop when it becomes sufficiently larger than $\|e_h + e_r - e_t\|$)

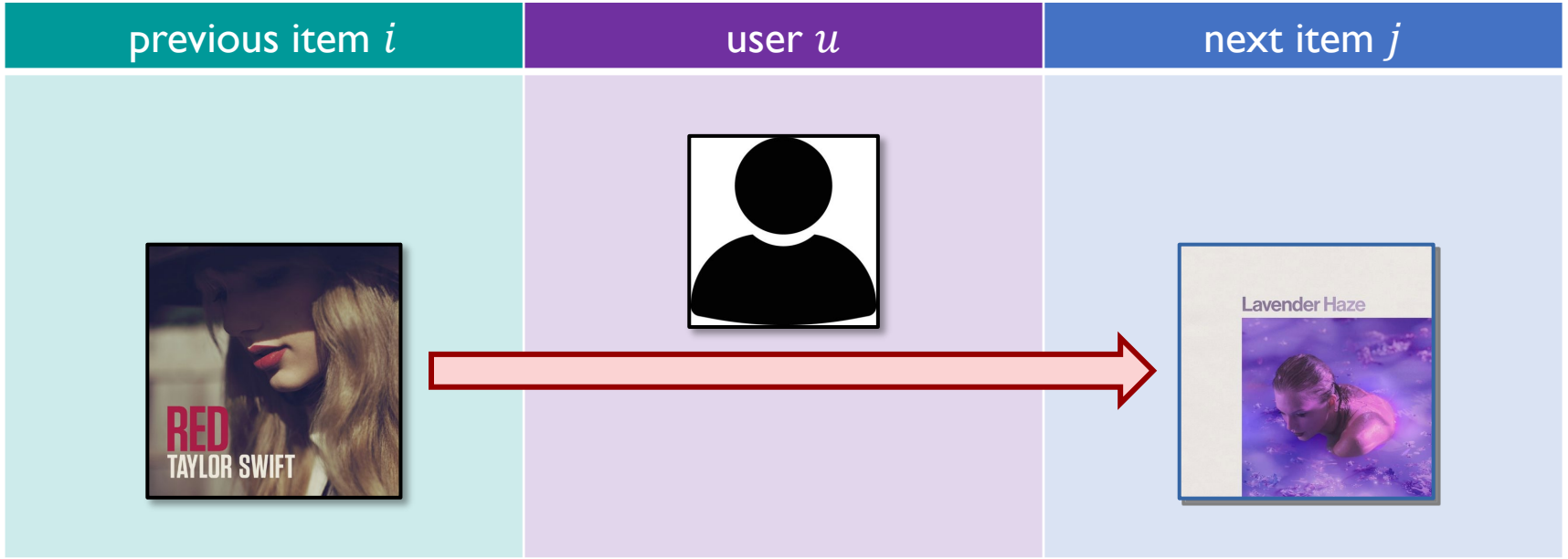
- Testing:

head entity h	relation r	tail entity t
Ottawa	is capital of	?

Find the nearest neighbor of $e_h + e_r$

Translation-Based Recommendation

- Items as points
- Users as translational vectors

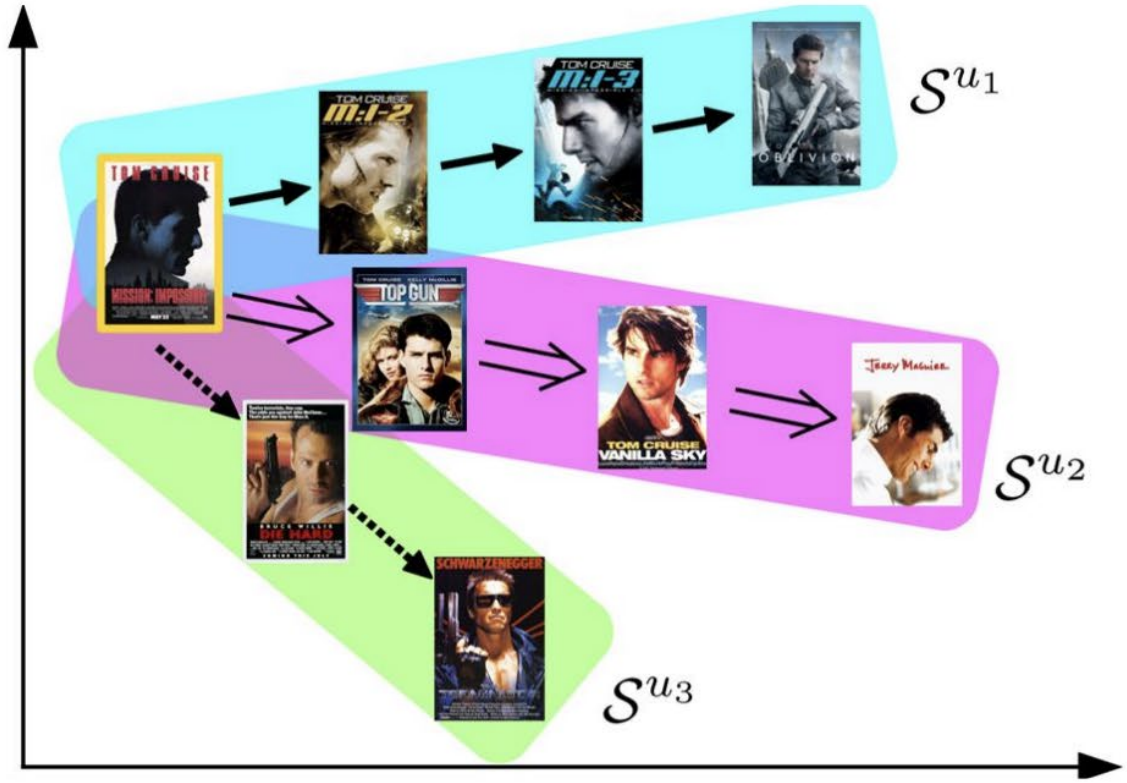


$$\boldsymbol{e}_i + \boldsymbol{e}_u \approx \boldsymbol{e}_j$$

Translation-Based Recommendation

- Items as points
- Users as translational vectors
  - E.g., the songs progressively become more up-tempo, as the user is at a party
    - $\boldsymbol{e}_u$:"the next song is more up-tempo than the previous song"
  - E.g., the songs gradually become more relaxing, as the user is going to sleep
    - $\boldsymbol{e}_u$:"the next song is more relaxing than the previous song"

Translation-Based Recommendation



User u_1

User u_2

User u_3

Translation operation:

$$\overrightarrow{prev.\ item} + \overrightarrow{user} \approx \overrightarrow{next\ item}$$

- Testing time: Find the nearest neighbor of $\boldsymbol{e}_i + \boldsymbol{e}_u$

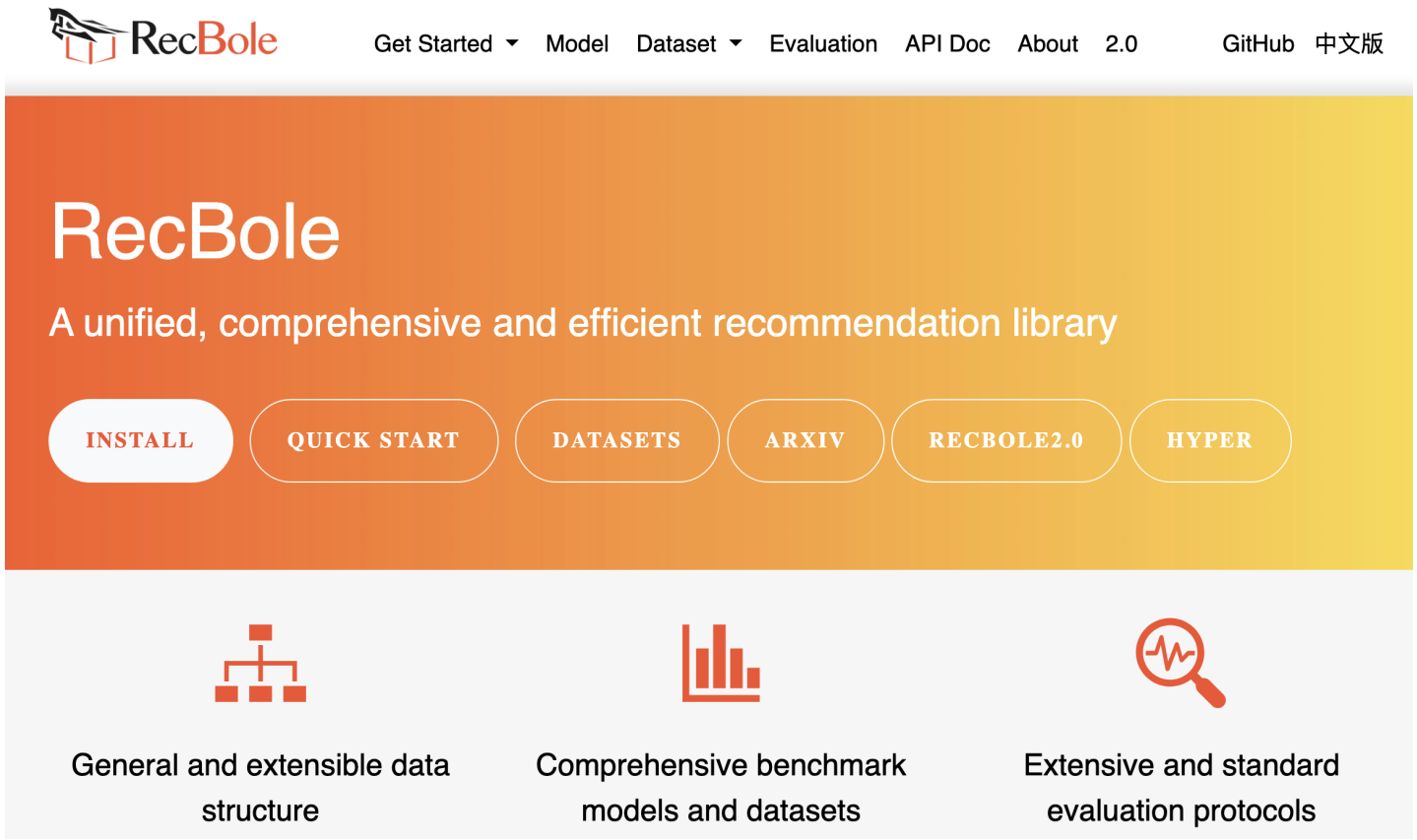
Performance

- Outperforms BPR-MF
- No comparisons with Fossil

Dataset	Metric	PopRec	BPR-MF	FMC	FPMC	HRM_{avg}	HRM_{max}	PRME	TransRec $\mathcal{L}_1$	TransRec $\mathcal{L}_2$	%Improv.
Epinions	*AUC*	0.4576	0.5523	0.5537	0.5517	0.6060	0.5617	0.6117	0.6063	0.6133	0.3%
	Hit@50	3.42%	3.70%	3.84%	2.93%	3.44%	2.79%	2.51%	3.18%	4.63%	20.6%
Automotive	*AUC*	0.5870	0.6342	0.6438	0.6427	0.6704	0.6556	0.6469	0.6779	0.6868	2.5%
	Hit@50	3.84%	3.80%	2.32%	3.11%	4.47%	3.71%	3.42%	5.07%	5.37%	20.1%
Google	*AUC*	0.5391	0.8188	0.7619	0.7740	0.8640	0.8102	0.8252	0.8359	0.8691	0.6%
	Hit@50	0.32%	4.27%	3.54%	3.99%	3.55%	4.59%	5.07%	6.37%	6.84%	32.5%
Office	*AUC*	0.6427	0.6979	0.6867	0.6866	0.6981	0.7005	0.7020	0.7186	0.7302	4.0%
	Hit@50	1.66%	4.09%	2.66%	2.97%	5.50%	4.17%	6.20%	6.86%	6.51%	10.7%
Toys	*AUC*	0.6240	0.7232	0.6645	0.7194	0.7579	0.7258	0.7261	0.7442	0.7590	0.2%
	Hit@50	1.69%	3.60%	1.55%	4.41%	5.25%	3.74%	4.80%	5.46%	5.44%	4.0%
Clothing	*AUC*	0.6189	0.6508	0.6640	0.6646	0.7057	0.6862	0.6886	0.7047	0.7243	2.6%
	Hit@50	1.11%	1.05%	0.57%	0.51%	1.70%	1.15%	1.00%	1.76%	2.12%	24.7%
Cellphone	*AUC*	0.6959	0.7569	0.7347	0.7375	0.7892	0.7654	0.7860	0.7988	0.8104	2.7%
	Hit@50	4.43%	5.15%	3.23%	2.81%	8.77%	6.32%	6.95%	9.46%	9.54%	8.8%
Games	*AUC*	0.7495	0.8517	0.8407	0.8523	0.8776	0.8566	0.8597	0.8711	0.8815	0.4%
	Hit@50	5.17%	10.93%	13.93%	12.29%	14.44%	12.86%	14.22%	16.61%	16.44%	15.0%
Electronics	*AUC*	0.7837	0.8096	0.8158	0.8082	0.8212	0.8148	0.8337	0.8457	0.8484	1.8%
	Hit@50	4.62%	2.98%	4.15%	2.82%	4.09%	2.59%	3.07%	4.89%	5.19%	12.3%
Foursquare	*AUC*	0.9168	0.9511	0.9463	0.9479	0.9559	0.9523	0.9565	0.9631	0.9651	0.9%
	Hit@50	55.60%	60.03%	63.00%	64.53%	60.75%	61.60%	65.32%	66.12%	67.09%	2.7%
Flixter	*AUC*	0.9459	0.9722	0.9568	0.9718	0.9695	0.9687	0.9728	0.9727	0.9750	0.2%
	Hit@50	11.92%	21.58%	22.23%	33.11%	32.34%	30.88%	40.81%	35.52%	35.02%	-13.0%

Homework 4 (Part 1): Compare Fossil and TransRec

- No need to implement them yourself!
- The RecBole library: https://recbole.io/



Thank You!

Course Website: https://yuzhang-teaching.github.io/CSCE670-S26.html