Information Retrieval

CS 547/DS 547

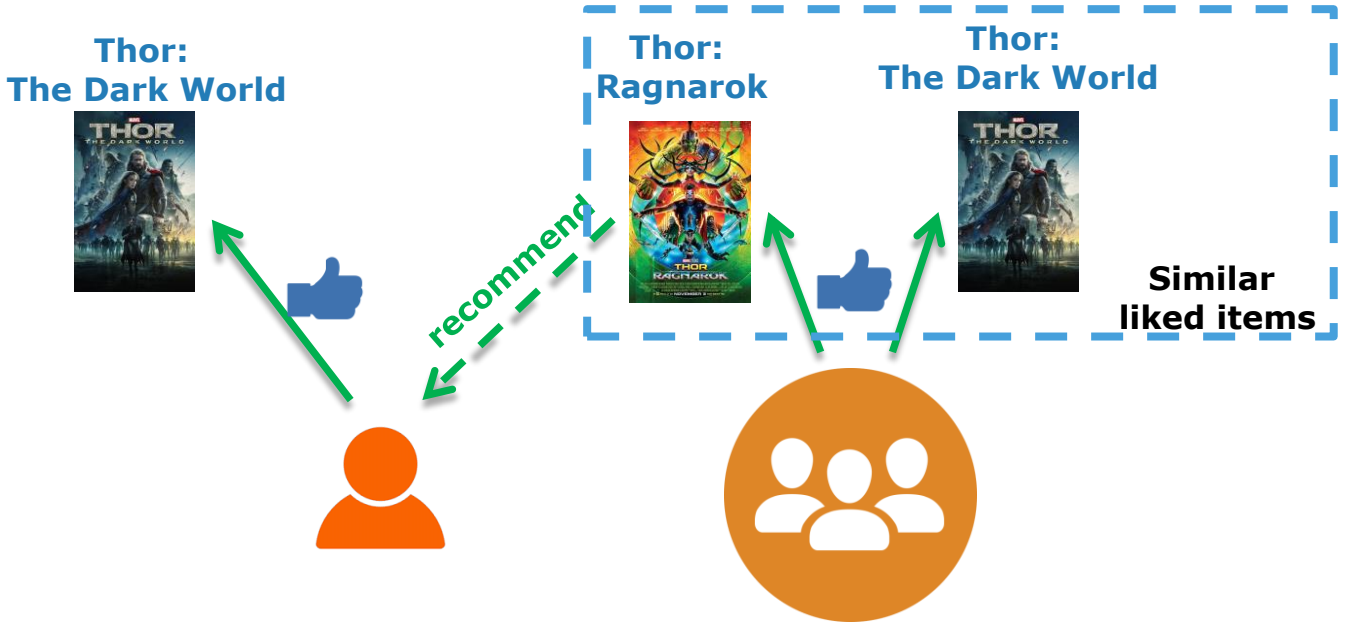
Worcester Polytechnic Institute
Department of Computer Science
Instructor: Prof. Kyumin Lee

Regularizing Matrix Factorization with User and Item Embeddings for Recommendation

T. Tran, K. Lee, Y. Liao, and D. Lee. Regularizing Matrix Factorization with User and Item Embeddings for Recommendation. CIKM, 2018.

Three observations in Recommender Systems: (1/3)

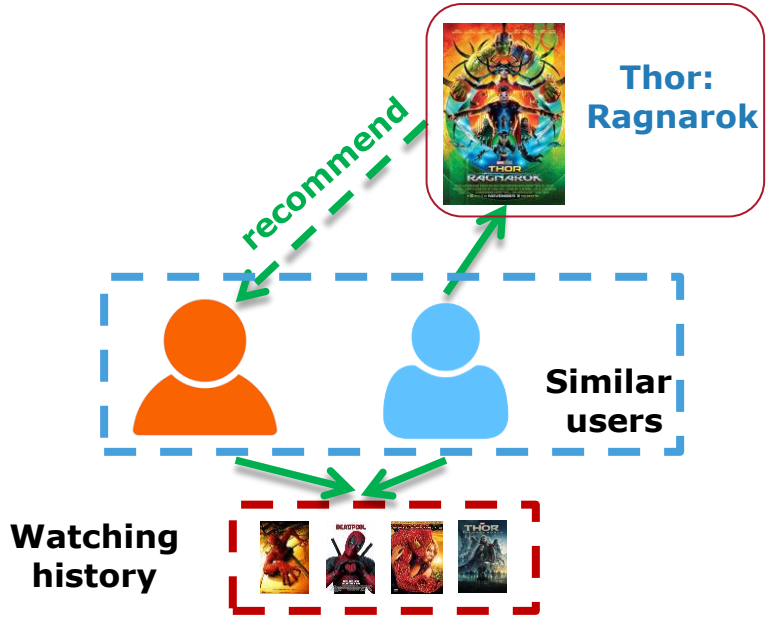
- **First observation**: recommend similar (co-liked) items to users.



Which pair of items is co-liked by how many people?

Three observations in Recommender Systems: (2/3)

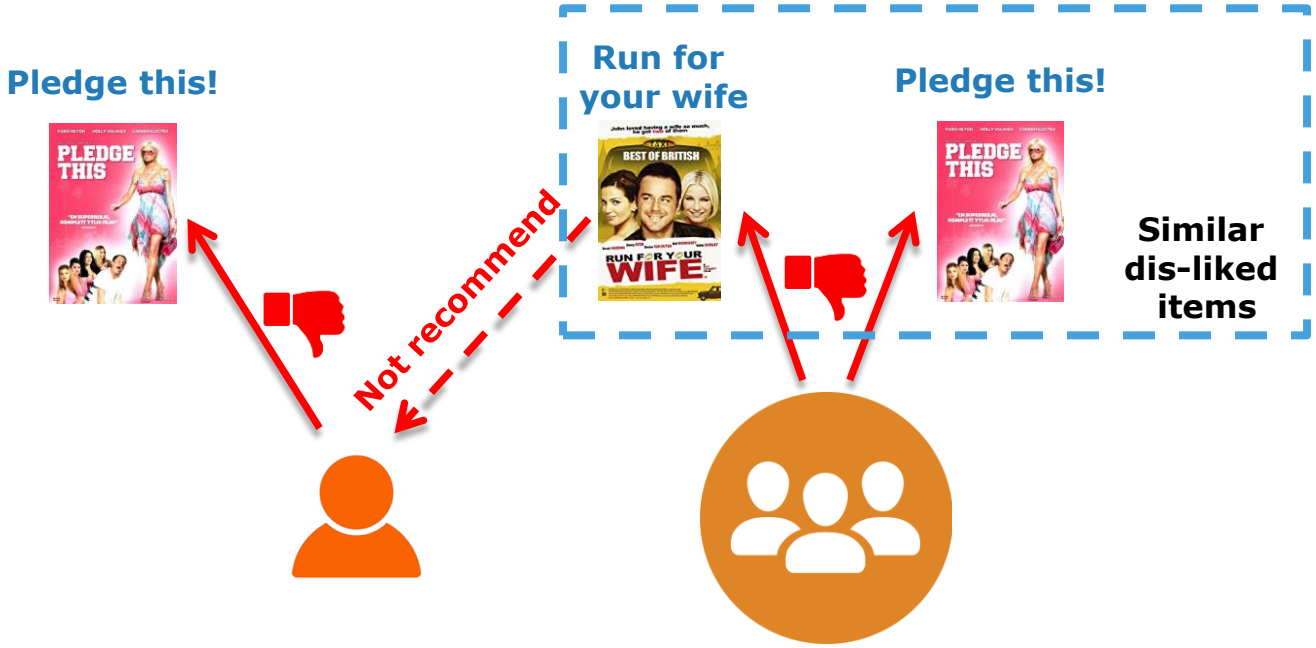
- **Second observation**: recommend same preferred items of a user to another similar user.



How many items does a pair of users liked together?

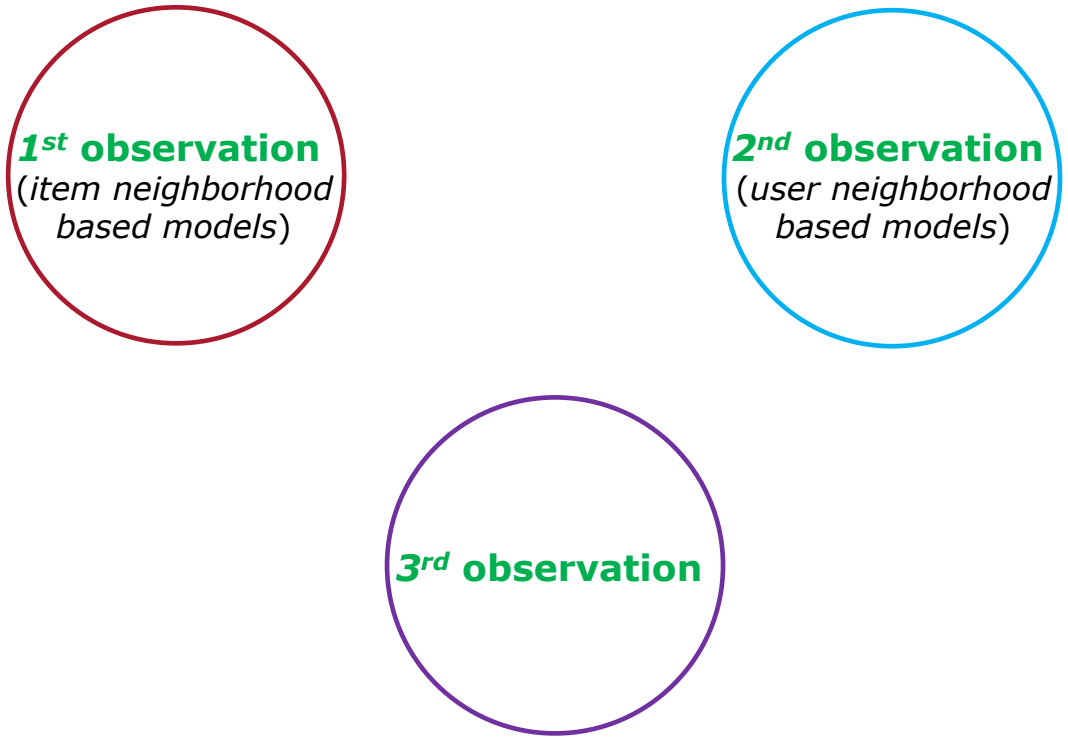
Three observations in Recommender Systems: (3/3)

- **Third observation**: not recommend similar disliked items to users.



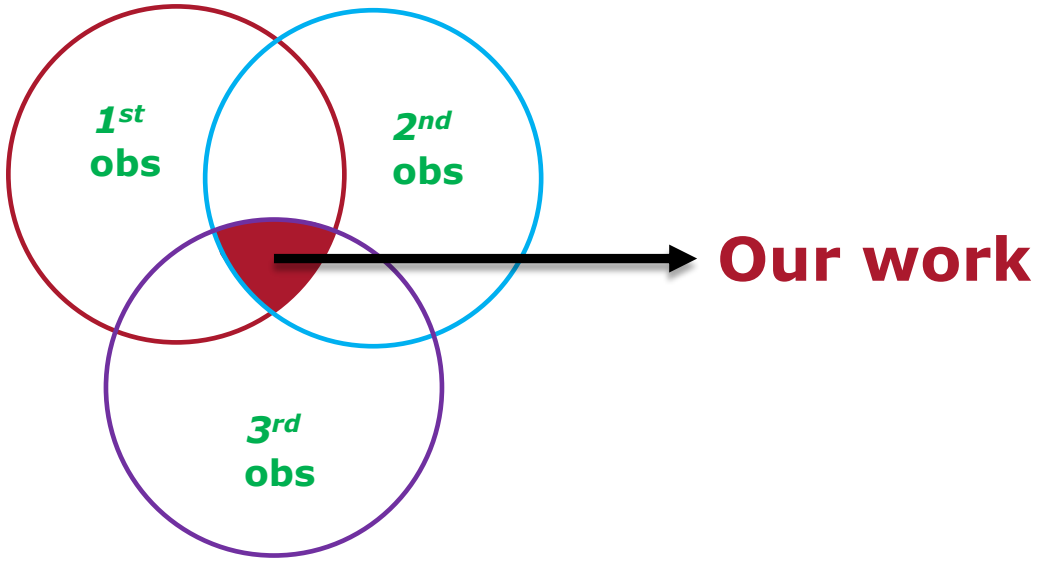
Which pair of items is co-disiked by how many people?

Our goal


1st observation
(item neighborhood based models)
2nd observation
(user neighborhood based models)
3rd observation

Our goal

- **Combine all:**



Research Questions

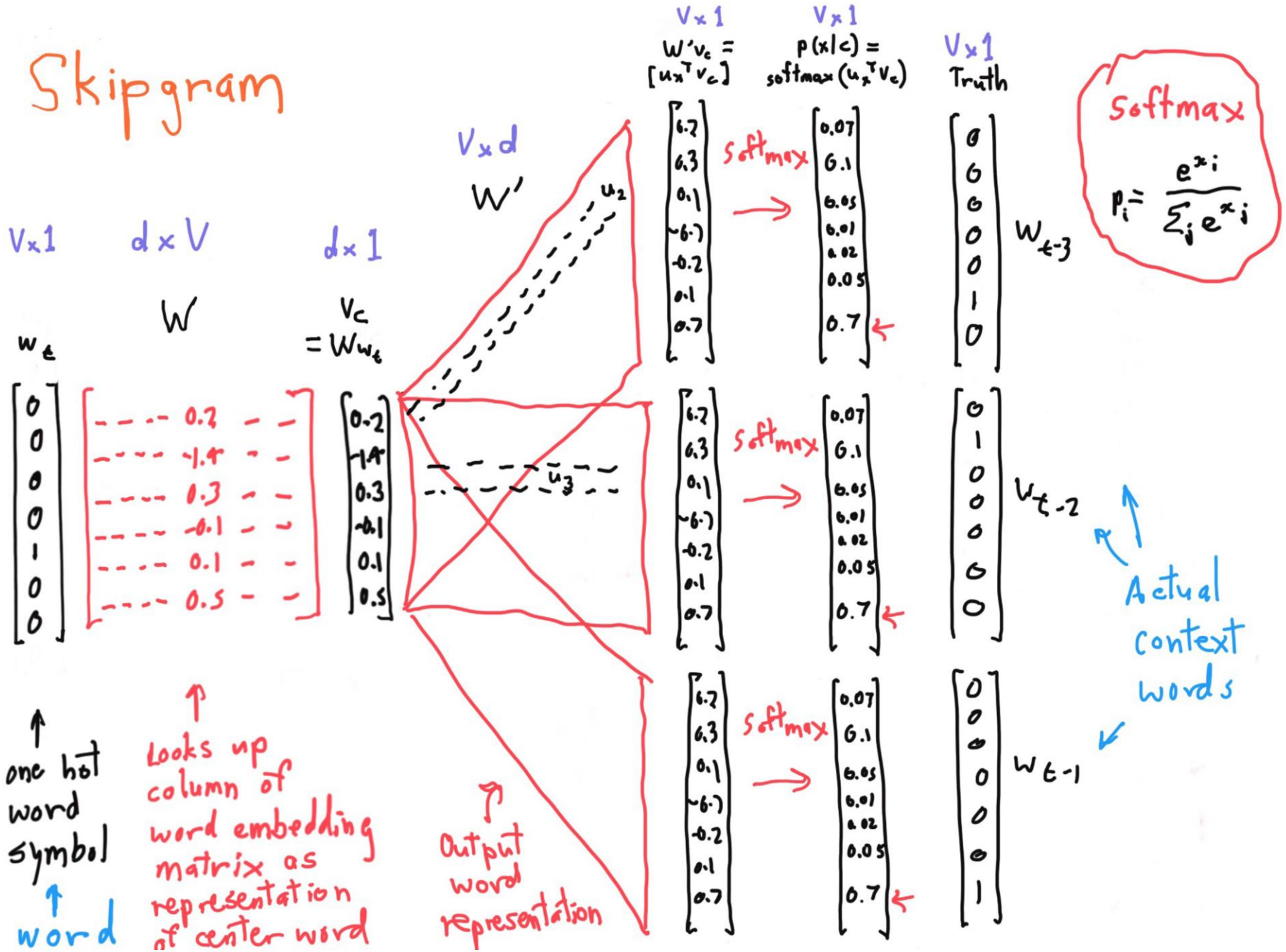
- **RS1:** How to exploit co-occurrence patterns of **users**, **co-liked items** and **co-disliked items**?
  - consider pairs of *co-occurred* **liked/disliked items** or pairs of *co-occurred* **users** as pairs of co-occurred words → apply word embedding technique.
- **RS2**: How to get disliked items in **implicit feedback** datasets?
  - Explicit feedback datasets: rating datasets (e.g. MovieLens), **visible disliked item** info.
  - Implicit feedback datasets: clicking datasets (e.g. TasteProfile), **no visible disliked item** info.

RQ1: Exploit co-occurrence patterns: skip-gram word2vec as implicit matrix factorization

- [Levy et al., 2014]: skip-gram word2vec is equivalent to implicitly factorizing a word-context matrix.

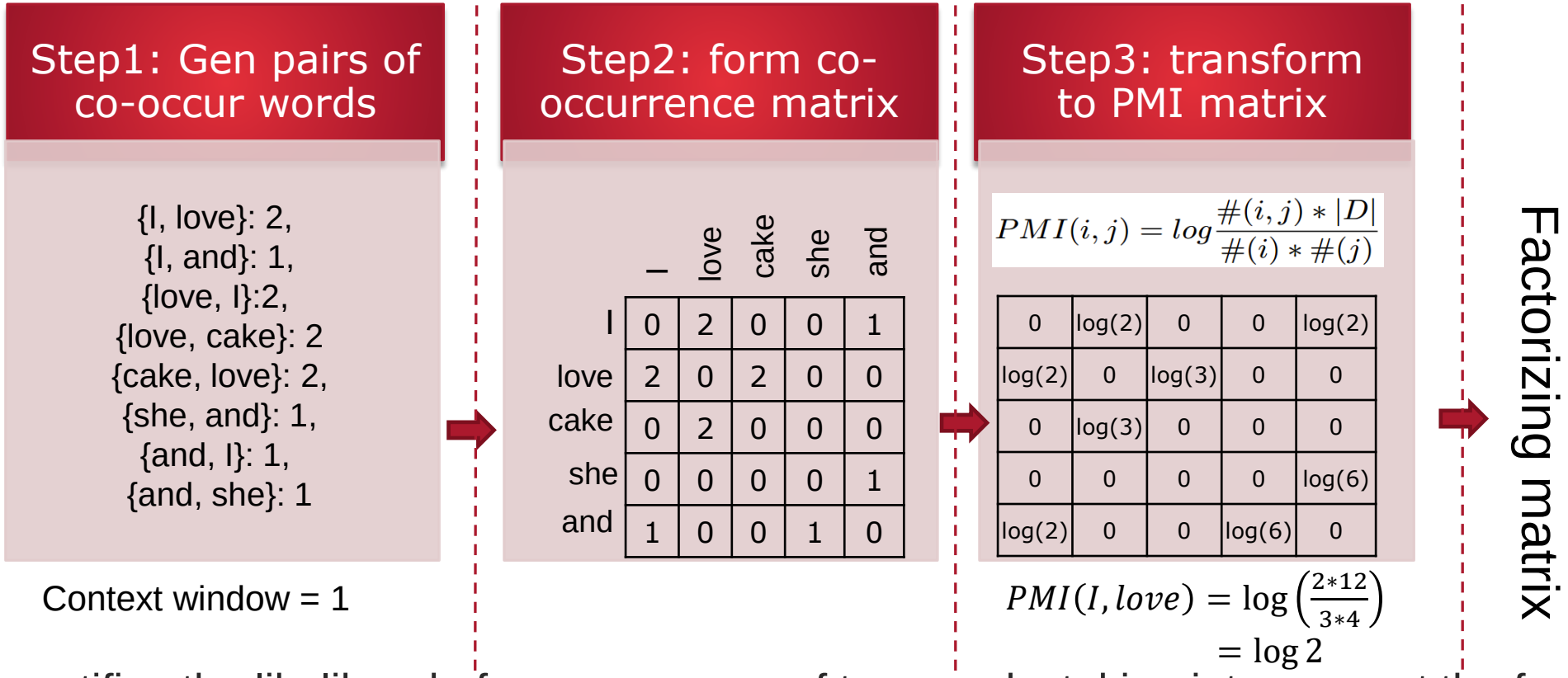
Neutral word embeddings as implicit matrix factorization, Levy & Goldberg, NIPS 2014

Skipgram


V x 1
W'v_c = [u_x^T v_c]
V x 1
p(x|c) = softmax(u_x^T v_c)
V x 1
Truth
V x d
W'
u_2
u_3
V x 1
d x V
d x 1
W
v_c = W w_t
w_t
Softmax
p_i = e^{x_i} / Σ_j e^{x_j}
softmax
softmax
softmax
W_{t-3}
W_{t-2}
W_{t-1}
Actual context words
one hot word symbol
word
Looks up column of word embedding matrix as representation of center word
Output word representation

RQ1: Exploit co-occurrence patterns: skip-gram word2vec as implicit matrix factorization

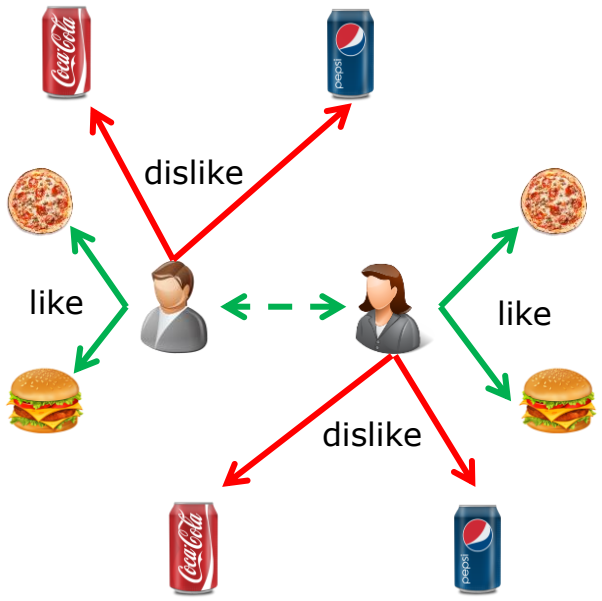
- Example: Given sentences: “I love cake”, “she and I love cake”.



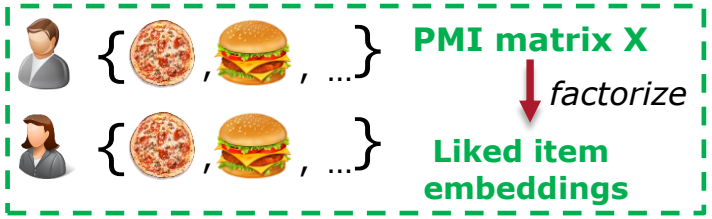
The PMI quantifies the likelihood of co-occurrence of two words, taking into account the fact that it might be caused by the frequency of the single words.

$$\text{pmi}(x;y) \equiv \log_2 \frac{p(x,y)}{p(x)p(y)} = \log_2 \frac{p(x|y)}{p(x)} = \log_2 \frac{p(y|x)}{p(y)}$$

RQ1: Problem mapping

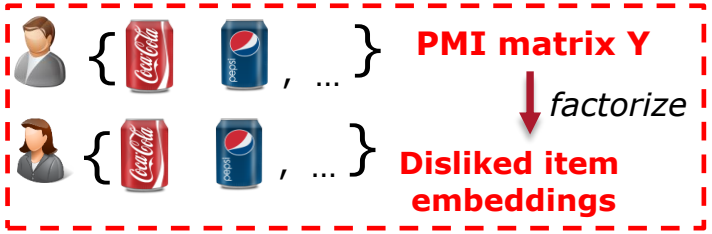


Which pair of items is co-liked by how many users?



How many items does a pair of users like together?

Which pair of items is co-disliked by how many users?



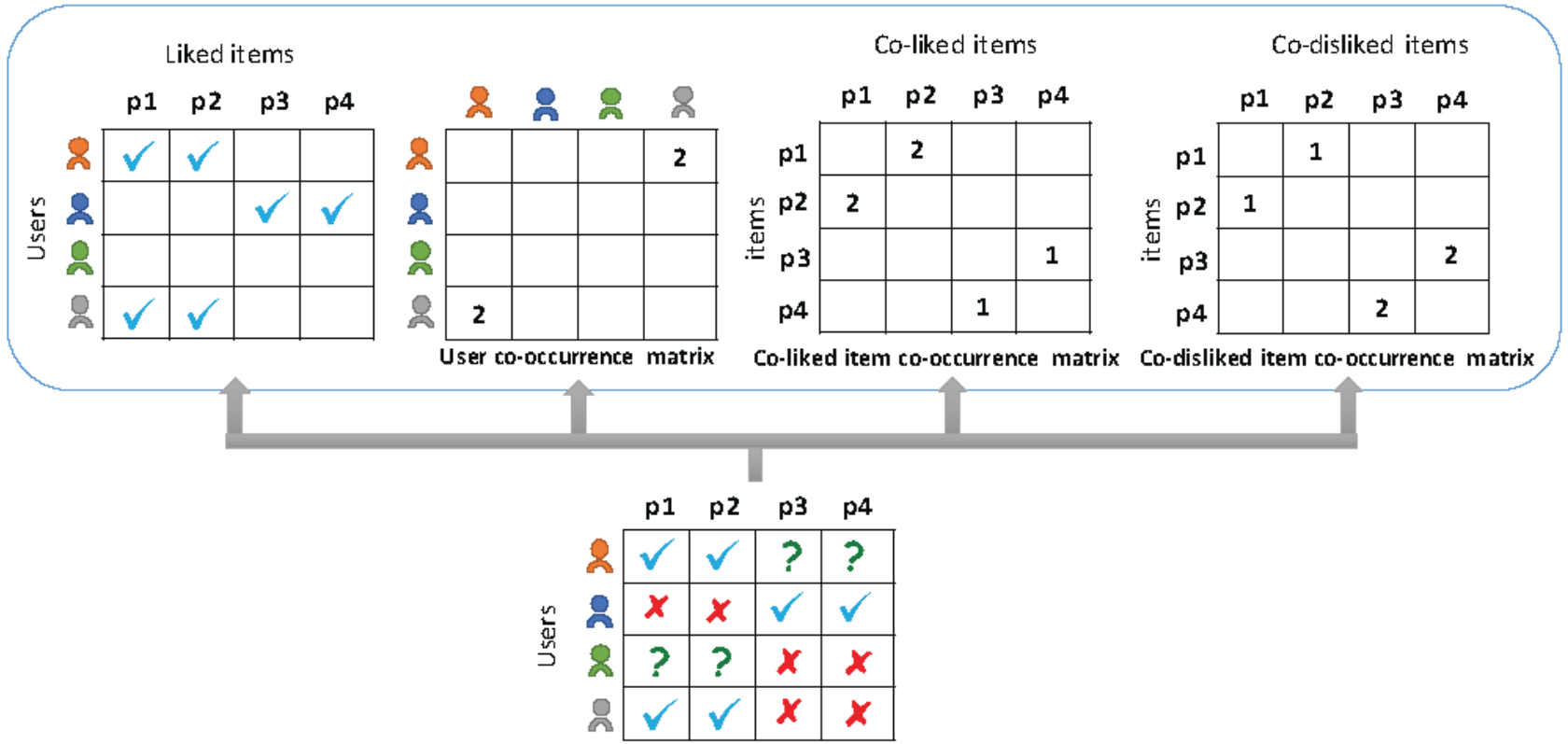


Figure 1: An overview of our RME Model, which jointly decomposes user-item interaction matrix, co-liked item co-occurrence matrix, co-disliked item co-occurrence matrix, and user co-occurrence matrix. (V: liked, X: disliked, and ?: unknown)

Our Regularized Multi-Embedding (RME) model

$$\textit{minimize}\ \mathrm{L} = \frac{1}{2}\sum_{u,p} w_{up}\left(M_{up} - \alpha_u^T \beta_p\right)^2$$

→ Squared loss of weighted matrix factorization

$$+\frac{1}{2}\sum_{X_{pi}\neq 0} w_{pi}\left(X_{pi} - \beta_p^T \gamma_i - b_p - c_i\right)^2$$

→ Liked item embeddings (1st obs)

$$+\frac{1}{2}\sum_{Y_{pj}\neq 0} w_{pj}\left(Y_{pj} - \beta_p^T \delta_j - d_p - e_j\right)^2$$

→ Disliked item embeddings (3rd obs)

$$+\frac{1}{2}\sum_{Z_{uk}\neq 0} w_{uk}\left(Z_{uk} - \alpha_u^T \theta_k - f_u - g_k\right)^2$$

→ User embeddings (2nd obs)

$$+\frac{1}{2}\lambda\left(\sum_u ||\alpha_u||^2 + \sum_p ||\beta_p||^2 + \sum_i ||\gamma_i||^2 + \sum_j ||\delta_j||^2 \sum_k ||\theta_k||^2\right)$$

Regularization

Our Regularized Multi-Embedding (RME) model

$$\textit{minimize}\ \mathrm{L} = \frac{1}{2}\sum_{u,p} w_{up}\left(M_{up} - \alpha_u^T \beta_p\right)^2$$
$$+\frac{1}{2}\sum_{X_{pi}\neq 0} w_{pi}\left(X_{pi} - \beta_p^T \gamma_i - b_p - c_i\right)^2$$
$$+\frac{1}{2}\sum_{Y_{pj}\neq 0} w_{pj}\left(Y_{pj} - \beta_p^T \delta_j - d_p - e_j\right)^2$$
$$+\frac{1}{2}\sum_{Z_{uk}\neq 0} w_{uk}\left(Z_{uk} - \alpha_u^T \theta_k - f_u - g_k\right)^2$$
$$+\frac{1}{2}\lambda\left(\sum_u ||\alpha_u||^2 + \sum_p ||\beta_p||^2 + \sum_i ||\gamma_i||^2 + \sum_j ||\delta_j||^2 \sum_k ||\theta_k||^2\right)$$

Shared item embeddings

Shared users embeddings

RQ2: RME model for implicit feedback dataset

- **How to get disliked items in implicit feedback datasets?**
- Uniform sampling
  - *[He et al. 2017]* --> **not good.**
- Item popularity awareness
  - [He et al. 2016] → **not personalized**.

→ design a **user-oriented EM-like algorithm** to sample disliked items for users.

Neural Collaborative Filtering, *He et al.*, WWW 2017.
Fast Matrix Factorization for online recommendation with implicit feedback, *He et al.*, SIGIR 2016.

RQ2: RME model for implicit feedback dataset

- Initialization: Extract **U, P** by running *WMF*(**M**) in the training set

- Iterate:
  - **E step:**
    - Estimate a probability distribution of items to be disliked by each user *u*.
      - Preference of user u on all items: $\boldsymbol{r^{(u)} = \alpha_u^T P}$
      - Probability of unobserved item ***i*** to be sampled as disliked item:

$$P_i^{(u)} = \frac{exp(-r_i^{(u)})}{\sum_{j=1}^{n} exp(-r_j^{(u)})}$$

    - Then, sample disliked items (= 0.2 * number of positive interactions per user *u)*.

  - **M step:** build *RME(M)* model, and then extract **U, P** from the model

- Terminate the loop until RME does not produce better recommendation results than the previous model in the validation dataset.

Experiment: Data Sets

- MovieLens-10M: movie rating dataset (5-star rating dataset).
- MovieLens-20M: movie rating dataset (5-star rating dataset).
- TasteProfile: users and songs with play count.

	MovieLens-10M	MovieLens-20M	TasteProfile
#users	58,057	111,146	221,011
#items	7,223	9,888	22,713
density	0.978%	0.745%	0.291%

Experiment: Compared methods

Baselines:

- WMF [1]: weighted low rank matrix factorization.
- Item-KNN [2]:an item neighborhood-based CF method.
- Item2Vec [3]: used SGNS to learn item embeddings, then apply Item-KNN to generate item recommendations.
- Cofactor [4]: combined WMF + liked item embeddings. (= **RME** - *user embeddings – disliked item embeddings).*

Two variants:

- U_RME: a variant of our model, **RME without** *disliked item embeddings*.
- I_RME: another variant of our model, **RME without** *user embeddings.*

[1] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering forimplicit feedback datasets. ICDM 2008.
[2] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. TOIS, 2004
[**3]** Oren Barkan and Noam Koenigstein. Item2vec: neural item embedding for collaborative filtering. MLSP, 2016
[4] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M Blei. Factor-ization meets the item embedding: Regularizing matrix factorization with itemco-occurrence. RecSys 2016.

Experiment: Metrics for Evaluation

- Split data: follow 70/10/20 proportion.
- Metrics:
  - Recall@N: consider ranking of first N items equivalent.
  - NDCG@N: normalized discount cumulative gain.
  - MAP@N: mean of user’s average precision (AP).

Experimental Research questions

- **RQ1**: Performance of **RME** versus baselines and its variants?
- **RQ2**: Hyper-parameter sensitivity analysis?
- **RQ3**: **RME**'s performance for different types of users (cold-start, warm-start, highly-active).

Experimental Results

- **RQ1**: **Performance of RME versus compared models?**
- **RQ2**: Hyper-parameter sensitivity analysis?
- **RQ3**: RME's performance for different types of users (cold-start, warm-start, highly-active).

RQ1: Performance of RME versus baselines (1/2)

- **MovieLens-20M dataset:**

	Item-KNN	Item2Vec	WMF	Cofactor	**U_RME**	**I_RME**	**RME**
Recall@5	0.0131	0.1066	0.1348	0.1480	0.1524	0.1530	**0.1570**
NDCG@20	0.0345	0.1019	0.1290	0.1387	0.1425	0.1412	**0.1461**
MAP@10	0.0402	0.0539	0.0720	0.0804	0.0847	0.0838	**0.0869**

our model's variants

Our model

RME improved: +6.5% on average over the best baseline, +2.6% on average over its two variants.

RQ1: Performance of RME versus baselines? (2/2)

- **TasteProfile dataset:**

	Item-KNN	Item2Vec	WMF	Cofactor	our model's variants: U_RME	our model's variants: I_RME	Our model: RME
Recall@5	0.0793	0.1455	0.1745	0.1771	0.1825	0.1826	**0.1876**
NDCG@20	0.0685	0.1593	0.1853	0.1873	0.1899	0.1915	**0.1954**
MAP@10	0.0904	0.0727	0.0931	0.095	0.0997	0.0996	**0.1025**

RME improved:+6.0% on average over the best baseline, +2.5% on average over its two variants.

Addition: RME with Implicit feedback Inference in Movielens dataset

- **MovieLens-20M dataset:**

	Item-KNN	Item2Vec	WMF	Cofactor	RME	RME_implicit
					Our model	
Recall@5	0.0131	0.1066	0.1348	0.1480	**0.1570**	**0.1547**
NDCG@20	0.0345	0.1019	0.1290	0.1387	**0.1461**	**0.1457**
MAP@10	0.0402	0.0539	0.0720	0.0804	**0.0869**	**0.0861**

RME_implicit: +5.6% on average over the best baseline, -0.9% on average over RME (with explicit dislike info)

Experimental Results:RQ2

- **RQ1**: Performance of MCF versus baselines?
- **RQ2: Hyper-parameter sensitivity analysis?**
  - **RQ 2.1: Varying top-N recommendation list? (refer to our paper)**
  - **RQ 2.2: Varying latent size K? (refer to our paper)**
  - **RQ 2.3: Varying negative sample ratio? (refer to our paper)**
  - **RQ 2.4: Dynamic settings of regularization hyper-parameters? (refer to our paper)**
- **RQ3**: Performance of MCF for different types of users (cold-start, warm-start, highly-active).

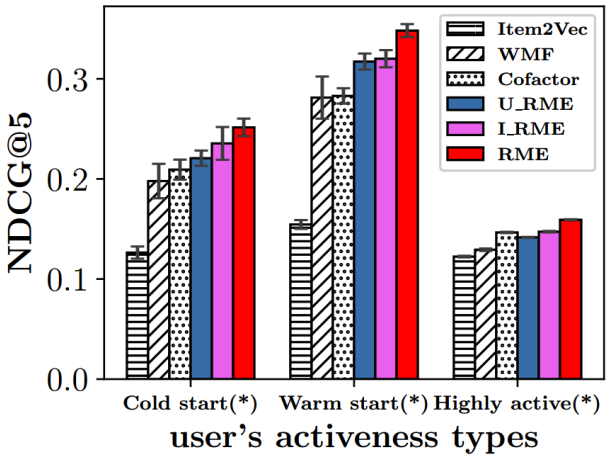
Experimental Results

- **RQ1**: Performance of RME versus baselines?
- **RQ2**: Hyper-parameter sensitivity analysis?
- **RQ3: RME's performance for different types of users (cold-start, warm-start, highly-active).**

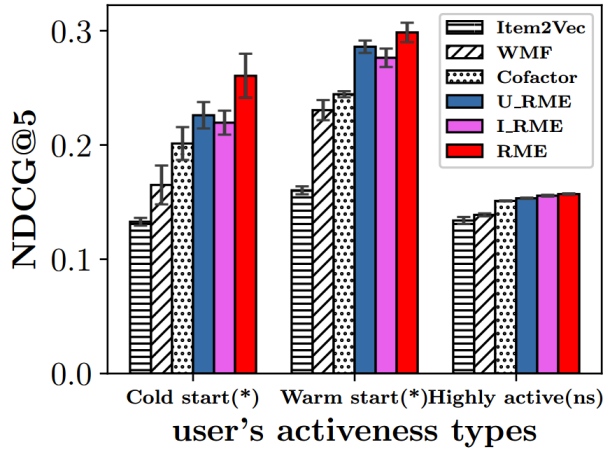
RQ3: Performance for different types of users

- Sort users by their activeness (e.g. #interactions) by ascending order.
- Cold-start: first 20%.
- Warm-start: 20%~80%.
- Highly active: last 20%.

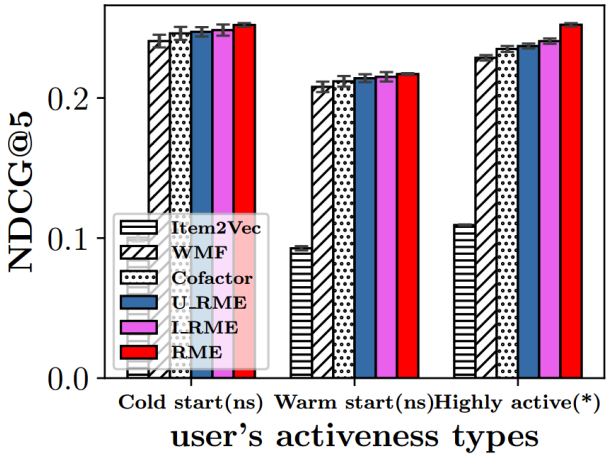
RQ3: Performance for different types of users



Movielens-10M



Movielens-20M



TasteProfile

On average, for cold-start users in two Movielens datasets: baseline: +24.8%; variants: +5.6%.

Summary

- Proposed a **joint model** combining WMF, co-liked item embedding, **co-disliked item** embedding and **user** embedding.
- Designed a **user-oriented EM-like** algorithm to sample **disliked items** for users.
- RME worked well on both explicit and implicit feedback datasets, **without using any additional info**: RME significantly improved **NDCG@20** by **6.6%** in all 3 datasets; RME improved **NDCG@5** by **24.8%** in two Movielens datasets for ***cold-start*** *users* group.

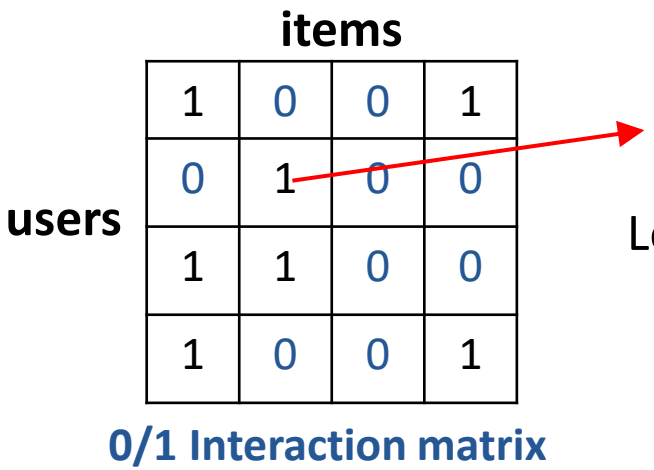
- Code: https://github.com/thanhdtran/RME.git

Neural Collaborative Filtering

Xiangnan et al. Neural Collaborative Filtering. WWW, 2017.

Matrix Factorization (MF)

- MF is a linear latent factor model:



User 'u' interacted with item 'i'

Learn latent vector for each user, item:

$$1 \times K$$

v_u^U ▭

v_i^I ▭

Affinity between user ‘u’ and item ‘i’:

$$\hat{y}_{ui} =< v_u, v_i >$$

Limitation of Matrix Factorization

- The simple choice of inner product function can limit the ***expressiveness*** of a MF model.

$$\hat{y}_{ui} = \mathbf{U}_i^T \mathbf{V}_j \simeq cos(\mathbf{U}_i, \mathbf{V}_j)$$ (E.g., assuming a unit length)

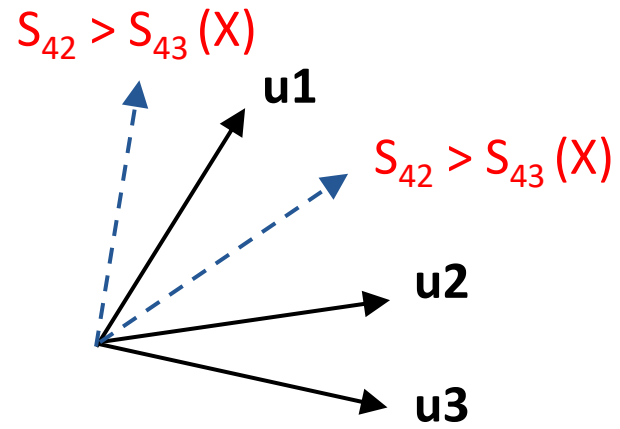
- Use Jaccard for similarity in this Example:

	i_1	i_2	i_3	i_4	i_5
u_1	1	1	1	0	1
u_2	0	1	1	0	0
u_3	0	1	1	1	0
u_4	1	0	1	1	1

sim(u1, u2) = 0.5

sim(u3, u1) = 0.4
sim(u3, u2) = 0.66

sim(u4, u1) = 0.6 *****
sim(u4, u2) = 0.2 *
sim(u4, u3) = 0.4 ***



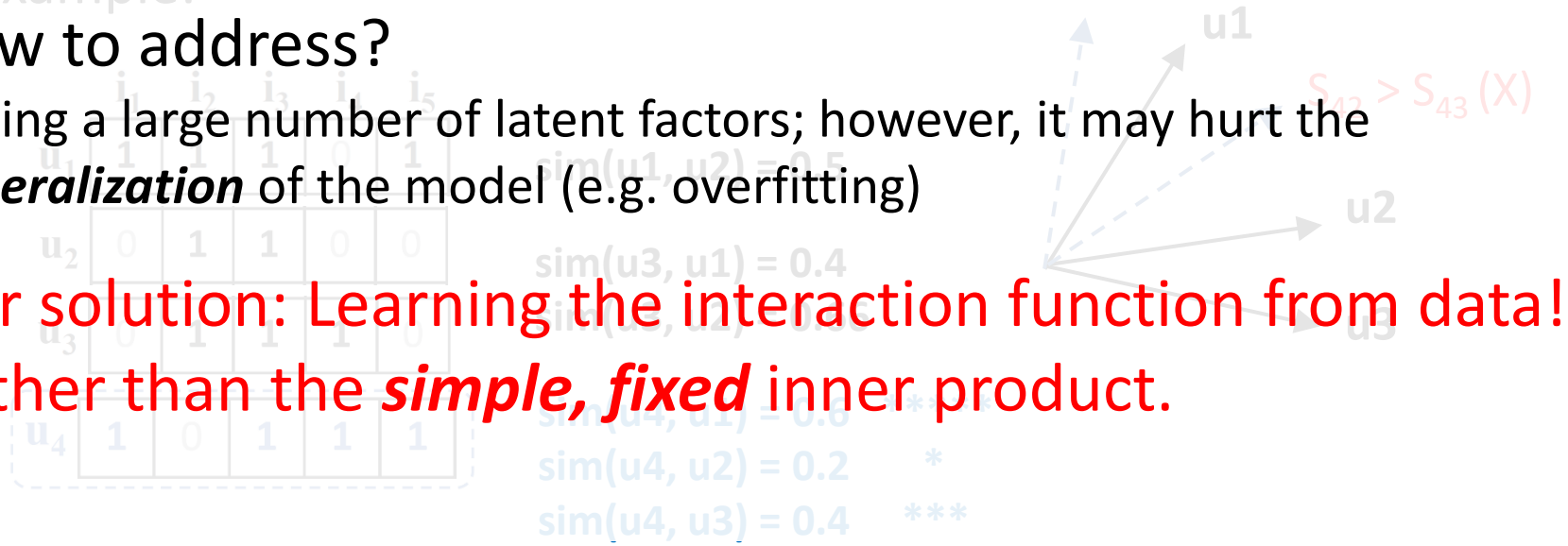
Jaccard Similarity: $s_{ij} = \frac{|\mathcal{R}_i| \cap |\mathcal{R}_j|}{|\mathcal{R}_i| \cup |\mathcal{R}_j|}$

Limitation of Matrix Factorization

- The simple choice of inner product function can limit the ***expressiveness*** of a MF model.

The inner product can incur a large **ranking** loss for MF

- Example:



How to address?

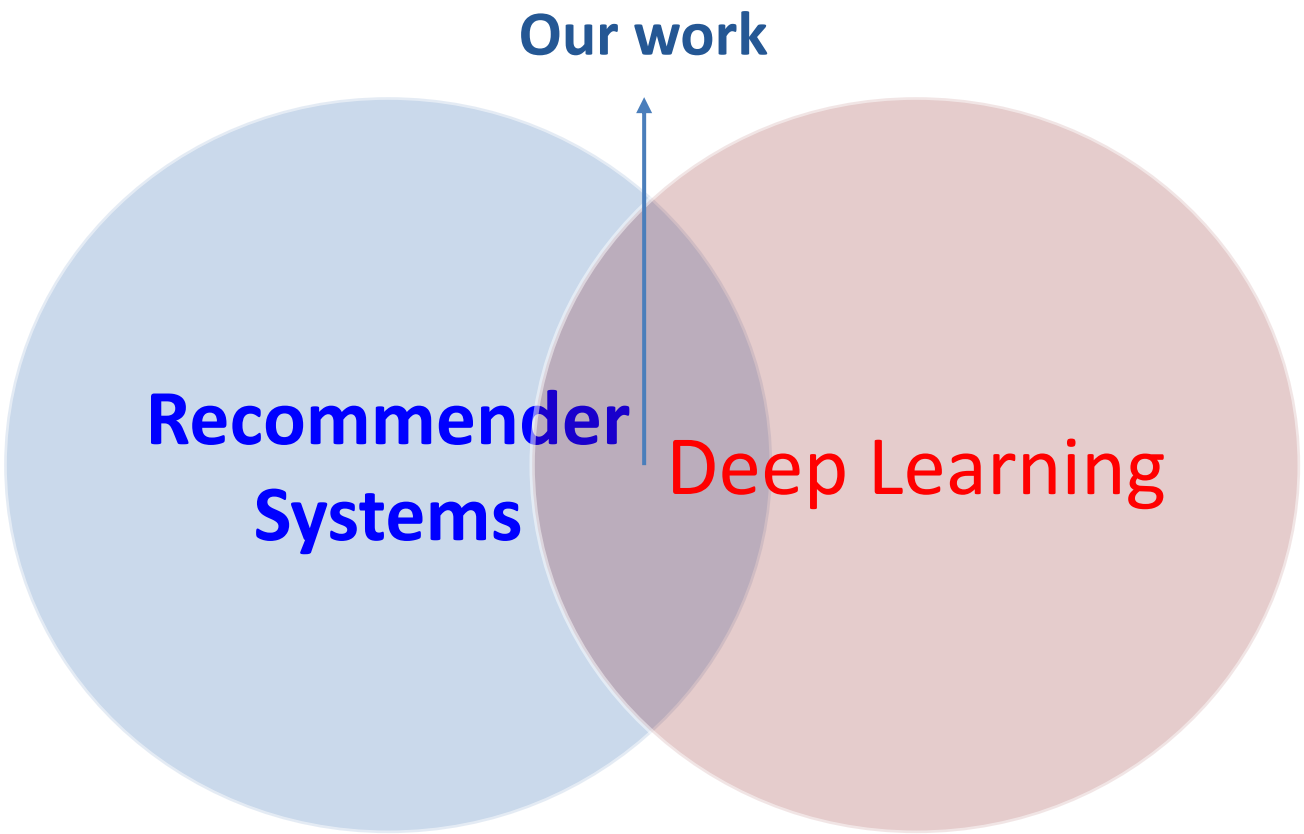
- Using a large number of latent factors; however, it may hurt the ***generalization*** of the model (e.g. overfitting)

Our solution: Learning the interaction function from data!

Rather than the ***simple, fixed*** inner product.

Jaccard Similarity: $s_{ij} = \frac{|\mathcal{R}_i| \cap |\mathcal{R}_j|}{|\mathcal{R}_i| \cup |\mathcal{R}_j|}$

Related Work



Related Work

- Zhang et al. KDD 2016. *Collaborative Knowledge Base Embedding for Recommender Systems*
- Y. Song et al. SIGIR 2016. *Multi-Rate Deep Learning for Temporal Recommendation*
- Li et al. CIKM 2015. *Deep Collaborative Filtering via Marginalized Denoising Auto-encoder*
- H. Wang et al. KDD 2015. *Collaborative Deep Learning for recommender systems*
- A. Elkahky et al. WWW 2015. *A Multi-View Deep Learning Approach for Cross Domain User Modeling in Recommendation Systems*
- X. Wang et al. MM 2014. *Improving content-based and hybrid music recommendation using deep learning*
- Oord et al. NIPS 2013. *Deep content-based music recommendation*

- Deep Learning (e.g., SDAE, CNN, SCAE) is only used for modelling SIDE INFORMATION of users and items.
- For modelling the interaction between users and items, existing work still uses the simple inner product

Proposed Methods

- Our Proposals:
  - A *Neural Collaborative Filtering* (NCF) framework that learns the interaction function with a deep neural network.
  - A NCF instance that generalizes the MF model (GMF).
  - A NCF instance that models nonlinearities with a multi-layer perceptron (MLP)
  - A NCF instance NeuMF that fuses GMF and MLP.

NCF Framework

Interaction function

$$\hat{y}_{ui} = f(\mathbf{p}_u, \mathbf{q}_i)$$

NCF uses a multi-layer model to learn the user-item interaction function

- Input: sparse feature vector for user *u* ($\mathbf{v}_u$) and item *i* ($\mathbf{v}_i$)
- Output: predicted score $\hat{y}_{ui}$

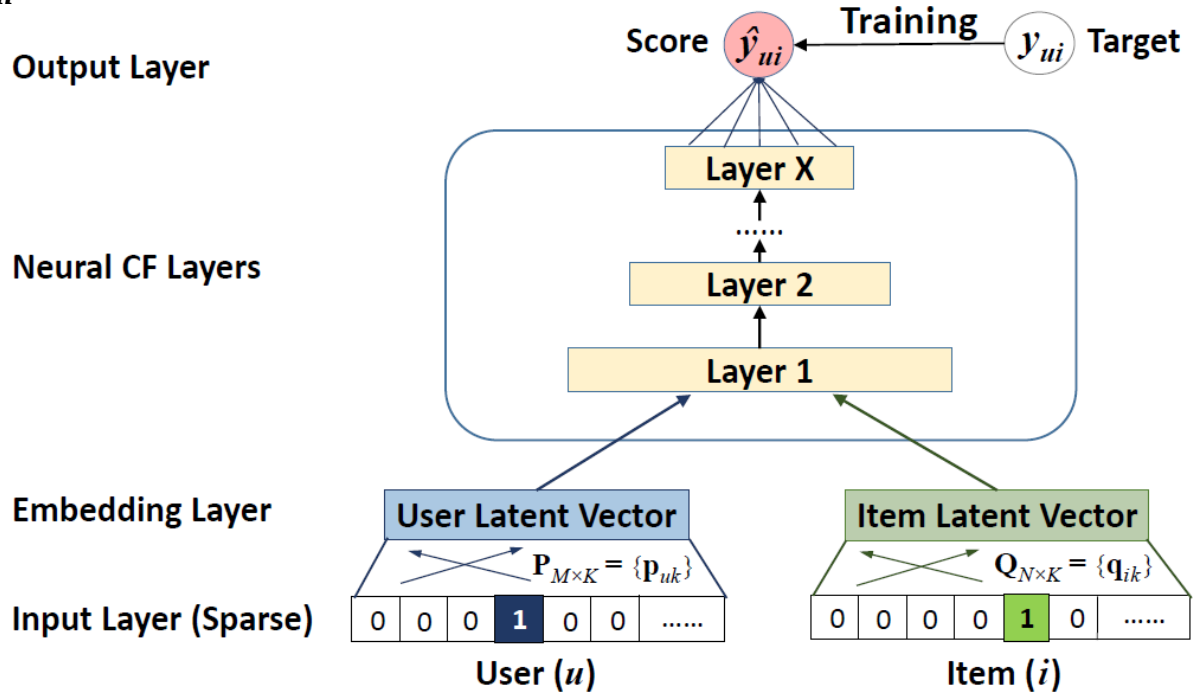


Figure 2: Neural collaborative filtering framework

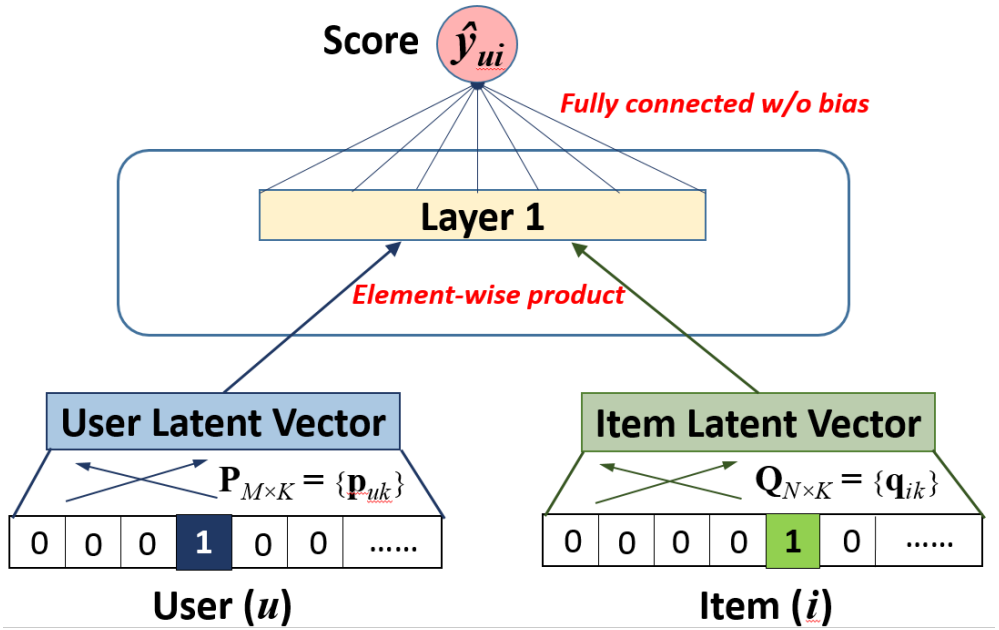
Generalized Matrix Factorization (GMF)

- NCF can express and generalize MF:

$$\hat{y}_{ui} = f(\mathbf{p}_u, \mathbf{q}_i)$$
$$= \phi_{out}(\phi_X(...\phi_2(\phi_1(\mathbf{p}_u, \mathbf{q}_i))...)),$$

Let we define *Layer 1* as an element-wise product, and *Output Layer* as a fully connected layer without bias, we have:

$$\hat{y}_{ui} = a_{out}(\mathbf{h}^T(\mathbf{p}_u \odot \mathbf{q}_i)),$$



Multi-Layer Perceptron (MLP)

- NCF can endow more nonlinearities to learn the interaction function:

Layer 1:

$$\mathbf{z}_1 = \phi_1(\mathbf{p}_u, \mathbf{q}_i) = \begin{bmatrix} \mathbf{p}_u \\ \mathbf{q}_i \end{bmatrix}$$

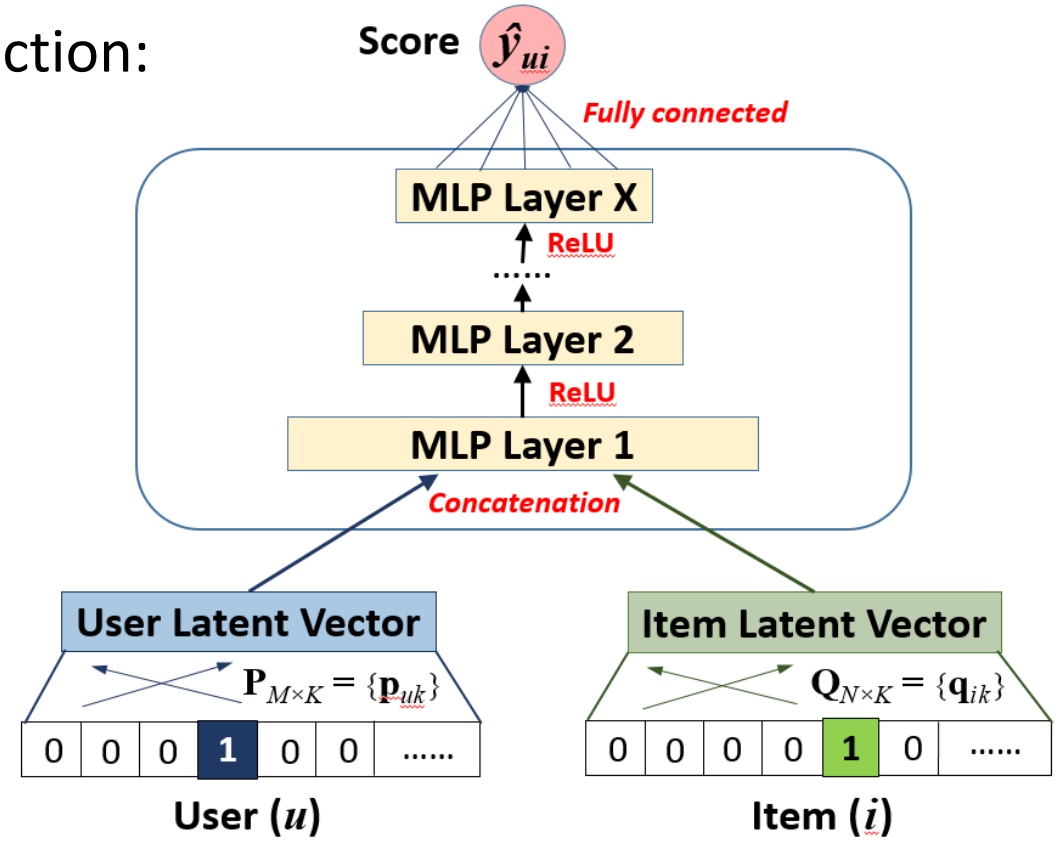
Remaining Layers:

$$\phi_2(\mathbf{z}_1) = a_2(\mathbf{W}_2^T \mathbf{z}_1 + \mathbf{b}_2),$$

$$\ldots\ldots$$

$$\phi_L(\mathbf{z}_{L-1}) = a_L(\mathbf{W}_L^T \mathbf{z}_{L-1} + \mathbf{b}_L),$$

$$\hat{y}_{ui} = \sigma(\mathbf{h}^T \phi_L(\mathbf{z}_{L-1})),$$



MF vs. MLP

$$\hat{y}_{ui} = f(\mathbf{p}_u, \mathbf{q}_i)$$

- MF uses an inner product as the interaction function:
  - Latent factors are independent with each other;
  - It empirically has good generalization ability for CF modelling

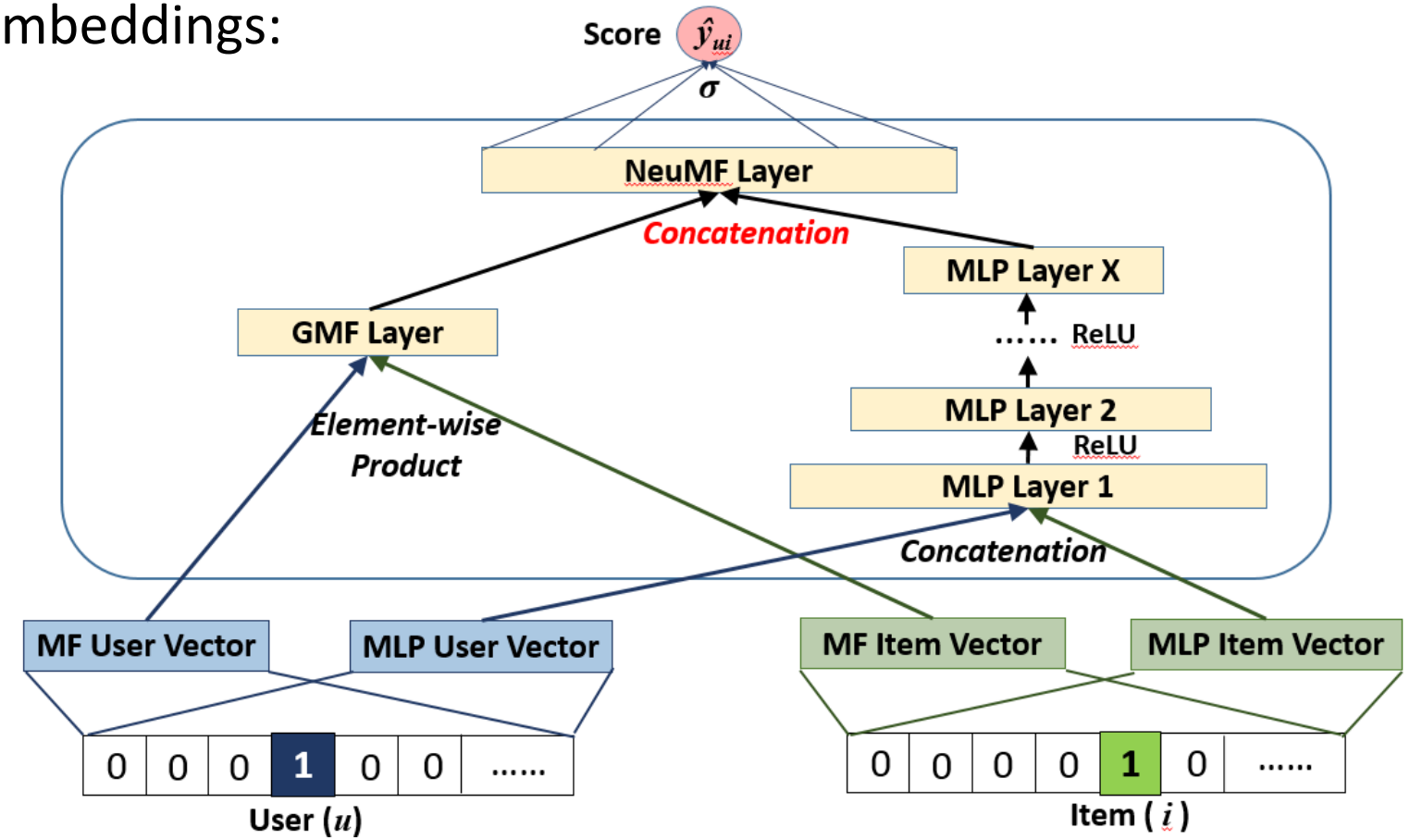
Can we fuse two models to get a more powerful model?

- MLP uses nonlinear functions to learn the interaction function:
  - Latent factors are correlated with each other
  - The interaction function is learnt from data, which conceptually has a better representation ability.
  - However, its generalization ability is unknown as it is seldom explored in recommender literature/challenge.


YES, WE CAN.

Our Fusion of GMF and MLP

- We propose a new *Neural Matrix Factorization* (NeuMF) model, which fuses GMF and MLP by allowing them learn different sets of embeddings:



Learning NCF Models

- For explicit feedback (e.g., ratings 1-5):

  Regression loss:

$$L_r = \sum_{(u,i)\in\mathcal{Y}} (y_{ui} - \hat{y}_{ui})^2 + \lambda_\Theta ||\Theta||^2$$

- For implicit feedback (e.g., watches, 0/1):

  Classification loss:

$$L_c = \sum_{(u,i)\in\mathcal{Y}\cup\mathcal{Y}^-} y_{ui} \log \hat{y}_{ui} + (1 - y_{ui}) \log(1 - \hat{y}_{ui}) + \lambda_\Theta ||\Theta||^2$$

- Optimization is done by SGD (*adaptive* learning rate variants: Adagrad, Adam, RMSprop…)

Experimental Setup

- Two public datasets from MovieLens and Pinterest:
  - Transform MovieLens ratings to 0/1 implicit case

Table 1: Statistics of the evaluation datasets.

Dataset	Interaction#	Item#	User#	Sparsity
MovieLens	1,000,209	3,706	6,040	95.53%
Pinterest	1,500,809	9,916	55,187	99.73%

- Evaluation protocols:
  - Leave-one-out: holdout the latest rating of each user as the test
  - Top-K evaluation
  - The ranked list are evaluated by Hit Ratio (i.e. recall) and NDCG (@10).

Baselines

- ItemPop.

  Items are ranked by their popularity.

- ItemKNN [Sarwar et al, WWW'01]

  The standard item-based CF method.

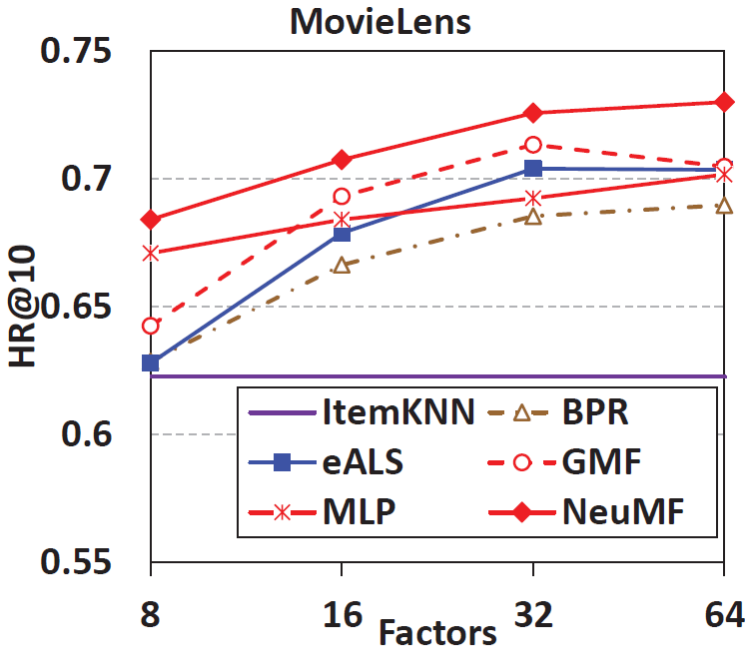
- BPR [Rendle et al, UAI'09]

  Bayesian Personalized Ranking optimizes MF model with a pairwise ranking loss, which is tailored for implicit feedback and item recommendation.

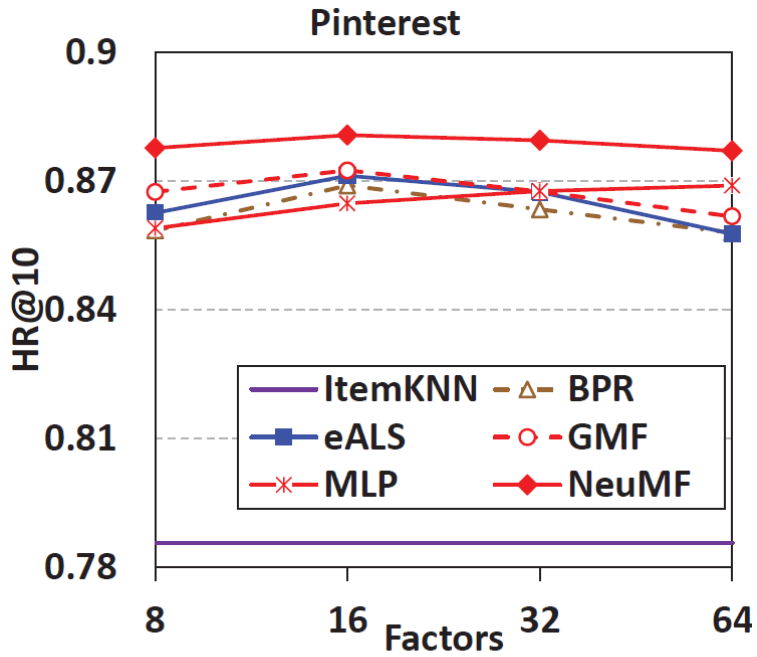
- eALS [He et al, SIGIR'16]

  The state-of-the-art CF method for implicit data. It optimizes MF model with a varying-weighted regression loss.

Performance vs. Embedding Size



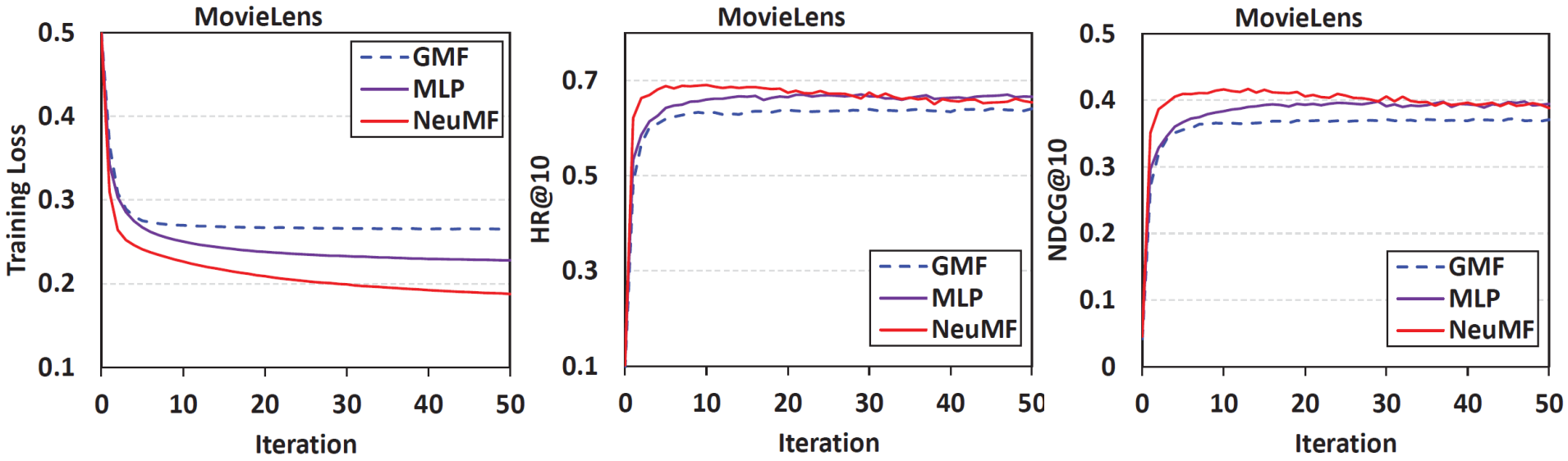
(a) MovieLens — HR@10



(c) Pinterest — HR@10

1. NeuMF outperforms eALS and BPR with about 5% relative improvement.
2. Of the three NCF methods: NeuMF > GMF > MLP (lower training loss but higher test loss)
3. Three MF methods with different objective functions:
   GMF (*log loss*) >= eALS (*weighted regression loss*) > BPR (*pairwise ranking loss*)

Convergence Behavior



(a) Training Loss (b) HR@10 (c) NDCG@10

1. Most effective updates are occurred in the first 10 iterations;
2. More iterations may make NeuMF overfit the data.

Trade-off between *representation* ability and *generalization* ability of a model.

Is Deeper Helpful?

Table 4: NDCG@10 of MLP with different layers.

Factors	MLP-0	MLP-1	MLP-2	MLP-3	MLP-4
MovieLens					
8	0.253	0.359	0.383	0.399	**0.406**
16	0.252	0.391	0.402	0.410	**0.415**
32	0.252	0.406	0.410	**0.425**	0.423
64	0.251	0.409	0.417	0.426	**0.432**
Pinterest					
8	0.141	0.526	0.534	0.536	**0.539**
16	0.141	0.532	0.536	0.538	**0.544**
32	0.142	0.537	0.538	0.542	**0.546**
64	0.141	0.538	0.542	0.545	**0.550**

1. Even for models with the same capability (i.e., same number of predictive factors), stacking more **nonlinear** layers improves the performance.
   - Note: stacking linear layers degrades the performance.
2. But the improvement gradually diminishes for more layers
   - Optimization difficulties (same observation with *K. He et al, CVPR 2016*)

Conclusion

- We explored neural architectures for collaborative filtering.
  - Devised a general framework NCF;
  - Presented three instantiations GMF, MLP and NeuMF.
- Experiments show promising results:
  - Deeper models are helpful.
  - Combining deep models with MF in the latent space leads to better results.

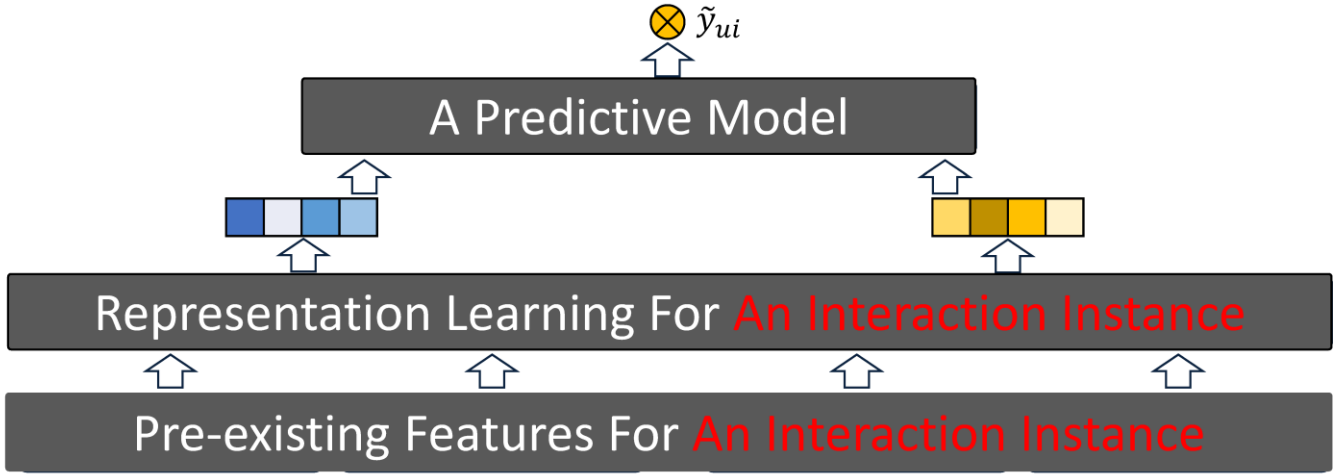
- Other resources:
  - https://towardsdatascience.com/paper-review-neural-collaborative-filtering-explanation-implementation-ea3e031b7f96
  - https://github.com/hexiangnan/neural_collaborative_filtering

Neural Graph Collaborative Filtering

Wang et al. Neural Graph Collaborative Filtering, SIGIR, 2019

Some of the following slides were adapted from the authors and Jure Leskovec

Modern Recommendation Paradigm



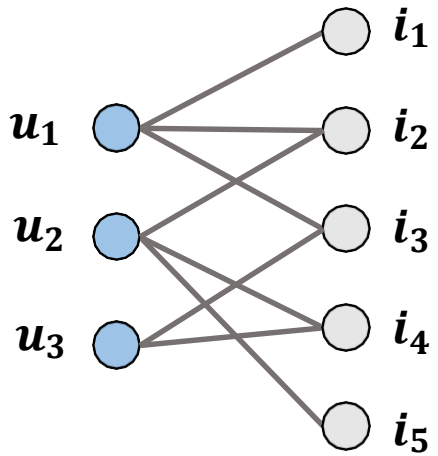
Information Isolated Island Effect:

- Model each instance individually
- While overlooking **relations** among instances
- Might result in **suboptimal performance**:
  - Making an instance's representation dependent only on its own features
    → Limited representation ability
  - Making interactions suffer from sparse issues
    → Suboptimal model capacity

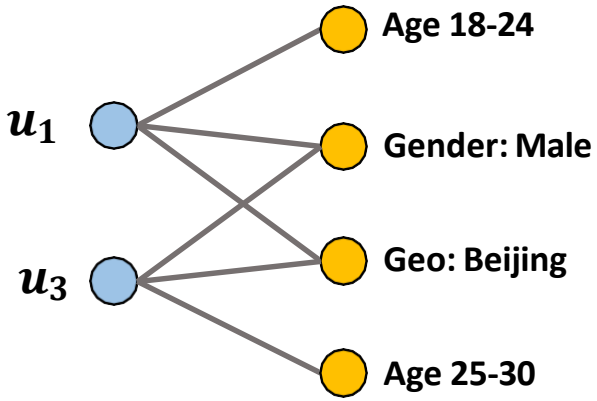
One Solution: Reorganizing Data into Graphs!

The data is more **closely connected** than we might think!

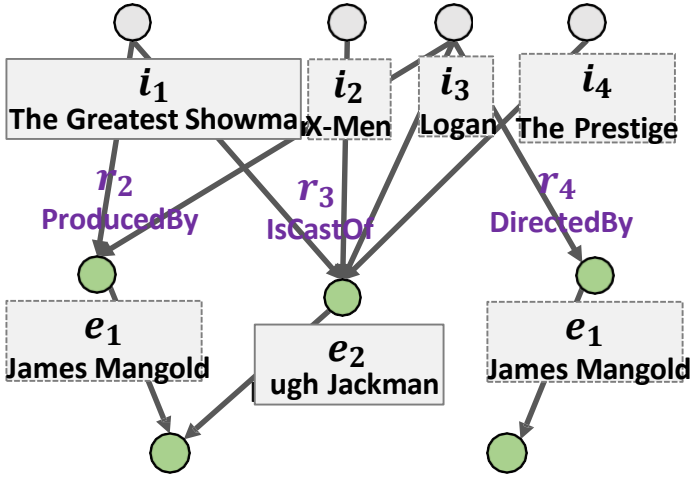
User-Item Interactions



User/Item Profiles



Knowledge Graph

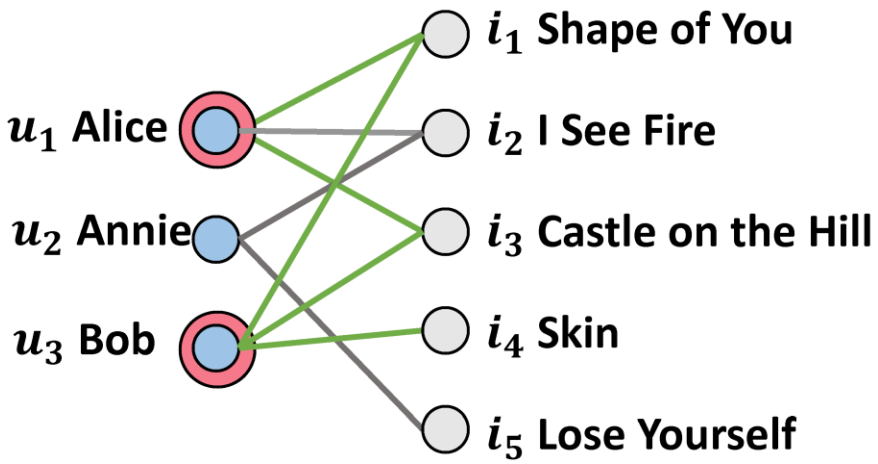


Limited Representation Ability ⇨ **Information Propagation along with the connections**

Collaborative Filtering (CF)

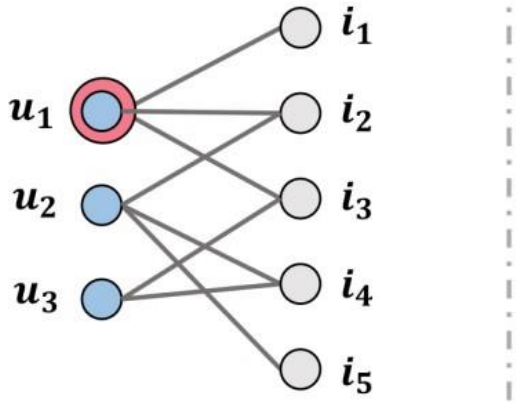
- **Collaborative Filtering** (CF) is the most well-known technique for recommendation.
  - Homophily assumption: a user preference can be predicted from his/her similar users.

- **Collaborative Signals -> Behavioral Similarity of users**
  - if u_1 and u_3 have interacted with the same items $\{i_1, i_3\}$, u_1 is likely to have similar preferences on other items $\{i_4\}$.

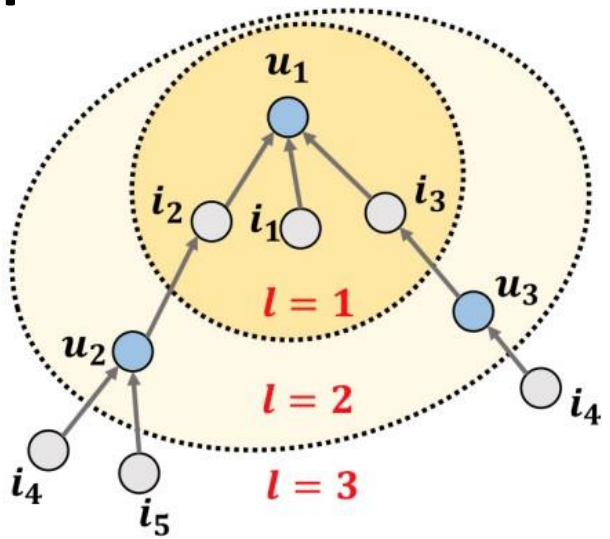


Revisiting CF via High-order Connectivity

- **High-order Connectivity from User-item Bipartite Graphs**
  - Why u_1 may like i_4?
    - $u_1 \leftarrow i_2 \leftarrow u_2 \leftarrow i_4$
    - $u_1 \leftarrow i_3 \leftarrow u_3 \leftarrow i_4$



User-Item Interaction Graph



High-order Connectivity for u_1

- Definition of high-order: the paths that reach u_1 from any node with the path length l larger than 1.

- A natural way to encode collaborative signals

Major contribution: CF modeling with high-order connectivity via GNN.

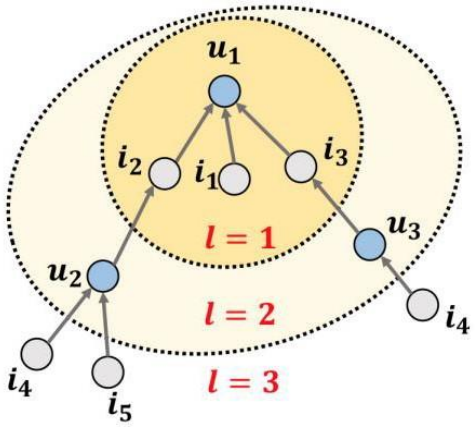
Neural Graph Collaborative Filtering

Embedding Propagation, inspired by GNNs

- Propagate embeddings recursively on the graph
- Construct information flows in the embedding space

➢ **First-order Propagation**

➢ **Message Construction:** generate message from one neighbor



message passed from i to u

$$\mathbf{m}_{u\leftarrow i} = \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}\left(\mathbf{W}_1\mathbf{e}_i + \mathbf{W}_2(\mathbf{e}_i \odot \mathbf{e}_u)\right)$$

discount factor

- message dependent on the affinity, distinct from GCN, GraphSage, etc.
- Pass more information to similar nodes

➢ **Message Aggregation:** update ego node's representation by aggregating message from all neighbors

$$\mathbf{e}_u^{(1)} = \text{LeakyReLU}\left(\mathbf{m}_{u\leftarrow u} + \sum_{i\in\mathcal{N}_u} \mathbf{m}_{u\leftarrow i}\right)$$

self-connections

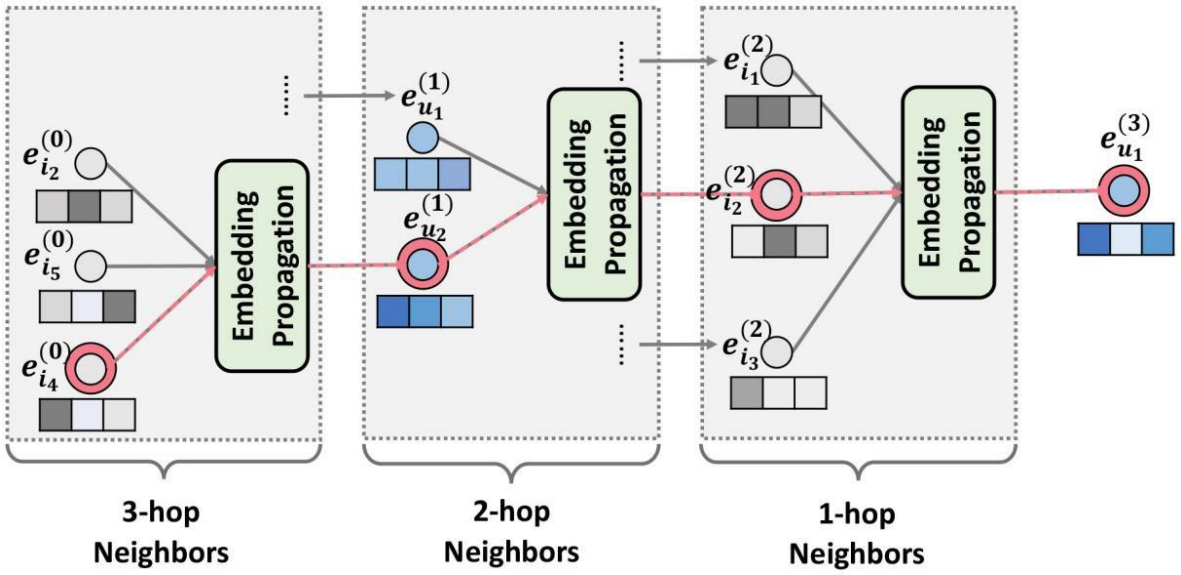
all neighbors of u

Neural Graph Collaborative Filtering

- **High-order Propagation**
  - We stack more embedding propagation layers to explore the high-order connectivity information.

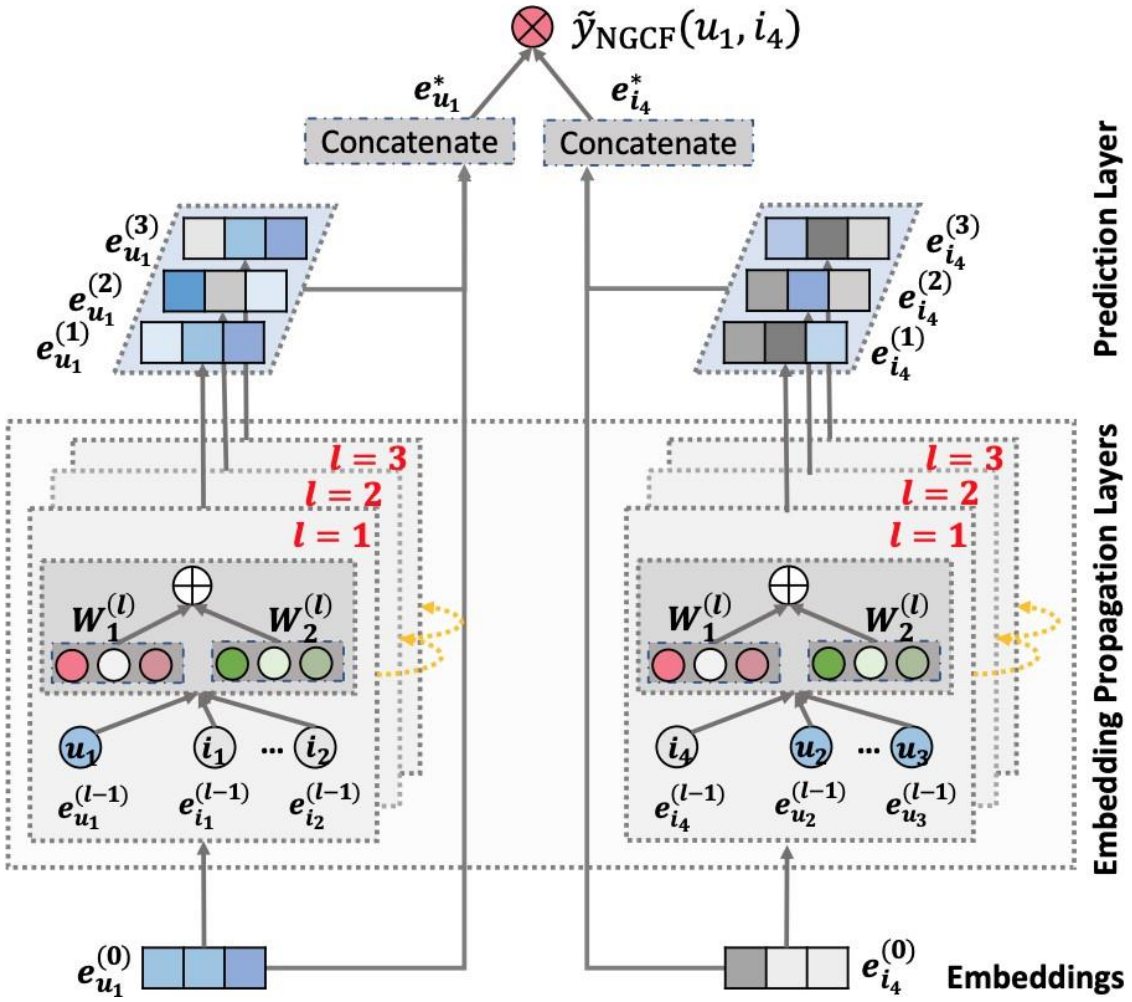
$$\mathbf{e}_u^{(l)} = \text{LeakyReLU}\left(\mathbf{m}_{u\leftarrow u}^{(l)} + \sum_{i\in\mathcal{N}_u} \mathbf{m}_{u\leftarrow i}^{(l)}\right),$$

representation of u at the l-th layer



- The collaborative signal like u1 ← i2 ← u2 ← i4 can be captured in the embedding propagation process.
- **Collaborative signal can be injected into the representation learning process.**

Overall Framework



Final embedding: concatenate the embedding from all layers

$$\mathbf{e}^*_u = \mathbf{e}^{(0)}_u \| \cdots \| \mathbf{e}^{(L)}_u$$

$$\mathbf{e}^*_i = \mathbf{e}^{(0)}_i \| \cdots \| \mathbf{e}^{(L)}_i$$

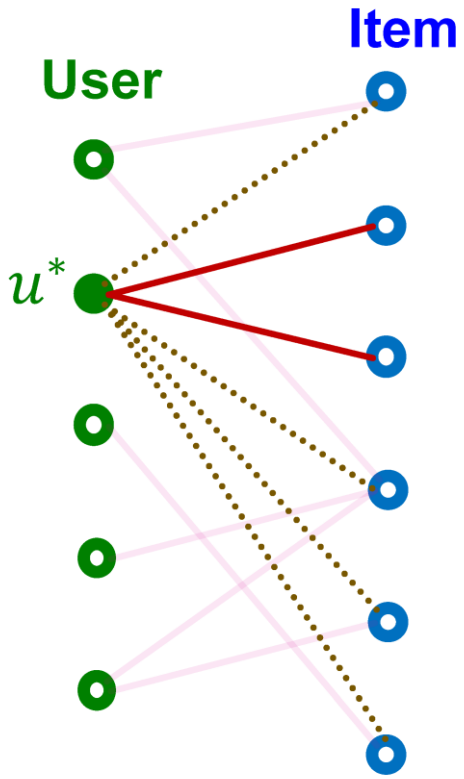
$$\hat{y}_{\text{NGCF}}(u, i) = {\mathbf{e}^*_u}^\top \mathbf{e}^*_i$$

The representations at different layers

- emphasize the messages passed over different connections
- have different contributions in reflecting user preference

Training: BPR Loss

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



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

Training: BPR Loss

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

Encouraged to be positive for each user
=positive edge score is higher than negative edge score

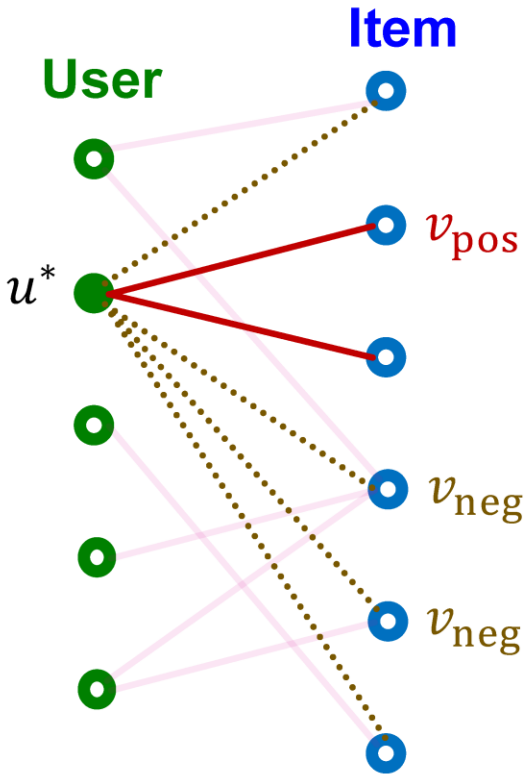
$$\text{Loss}(u^*) = \frac{1}{|E(u^*)| \cdot |\boldsymbol{E}_{\mathbf{neg}}(u^*)|} \underbrace{\sum_{(u^*, v_{\text{pos}}) \in \boldsymbol{E}(u^*)} \sum_{(u^*, v_{\text{neg}}) \in \boldsymbol{E}_{\mathbf{neg}}(u^*)} -\log\left(\sigma\left(f_\theta(\boldsymbol{u}^*, \boldsymbol{v}_{\text{pos}}) - f_\theta(\boldsymbol{u}^*, \boldsymbol{v}_{\text{neg}})\right)\right)}$$

Can be approximated using a mini-batch

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

Training: BPR Loss

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



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

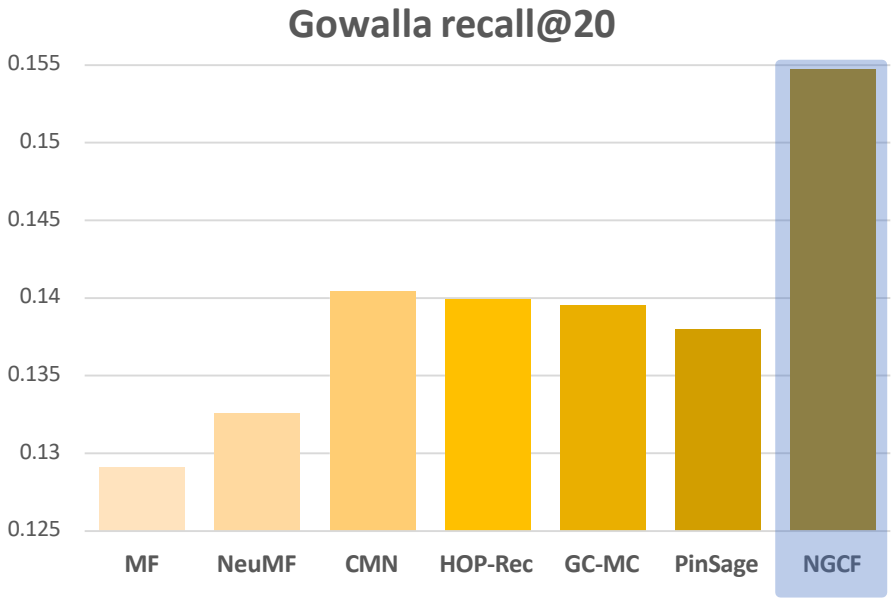
Average over users in the mini-batch

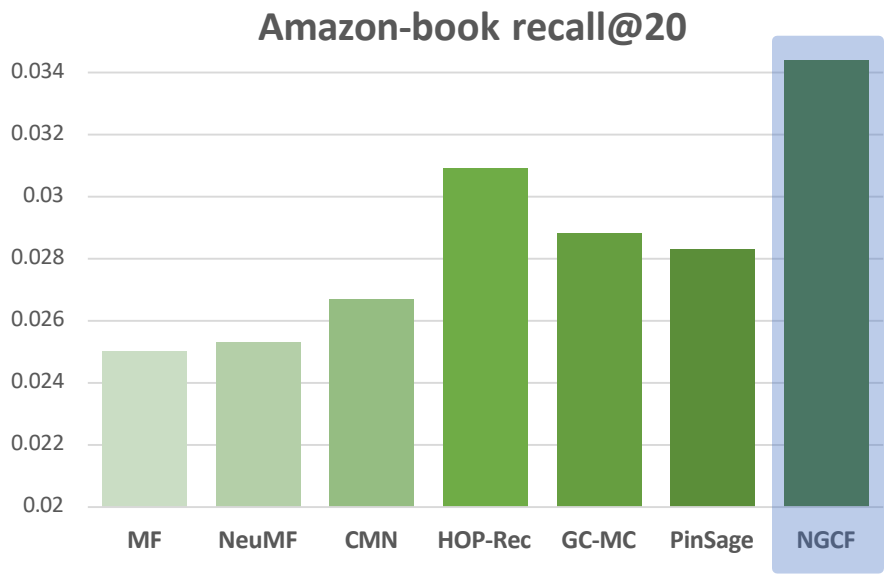
Experiments

- **Datasets**
  - Gowalla, Amazon-Book, Yelp2018
- **Evaluation Metrics**
  - recall@K, ndcg@K
- **Baselines**

	Data for Embedding Function	Connectivity	Aggregation Type in GNNs
MF	ID	-	-
NeuFM	ID	-	-
CMN	Personal History	First-order	-
HOP-Rec	Multi-hop Neighbors	High-order	-
PinSage	Collaborative Signals	Second-order	Concatenation
GC-MC	Collaborative Signals	First-order	Sum
NGCF	Collaborative Signals	**High-order**	Sum + **Element-wise Product**

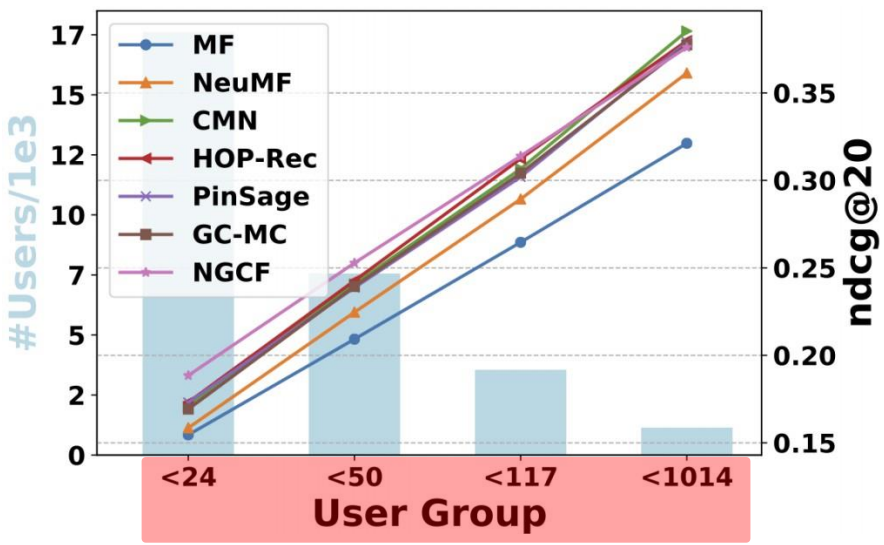
Overall Performance Comparison (1)

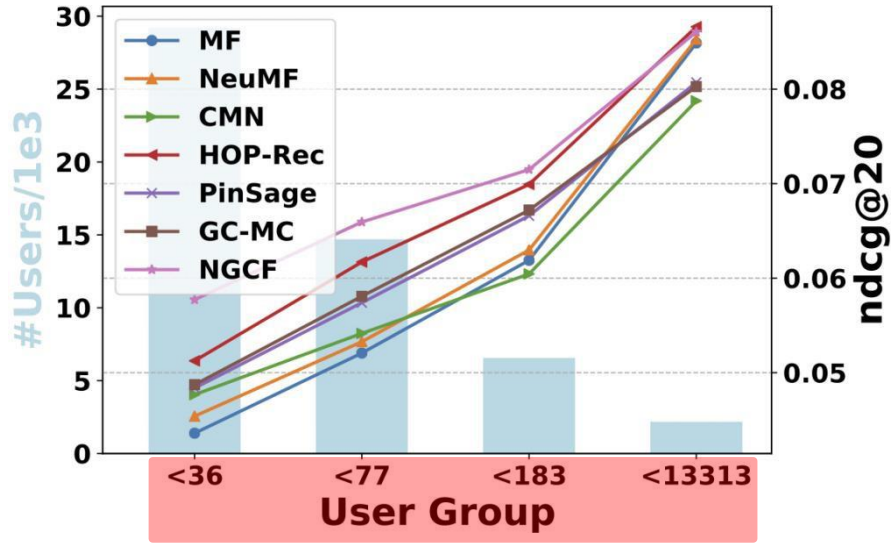




- NGCF consistently yields the best performance on all the datasets.
- **This verifies the importance of capturing collaborative signal in embedding function.**

Overall Performance Comparison (2)

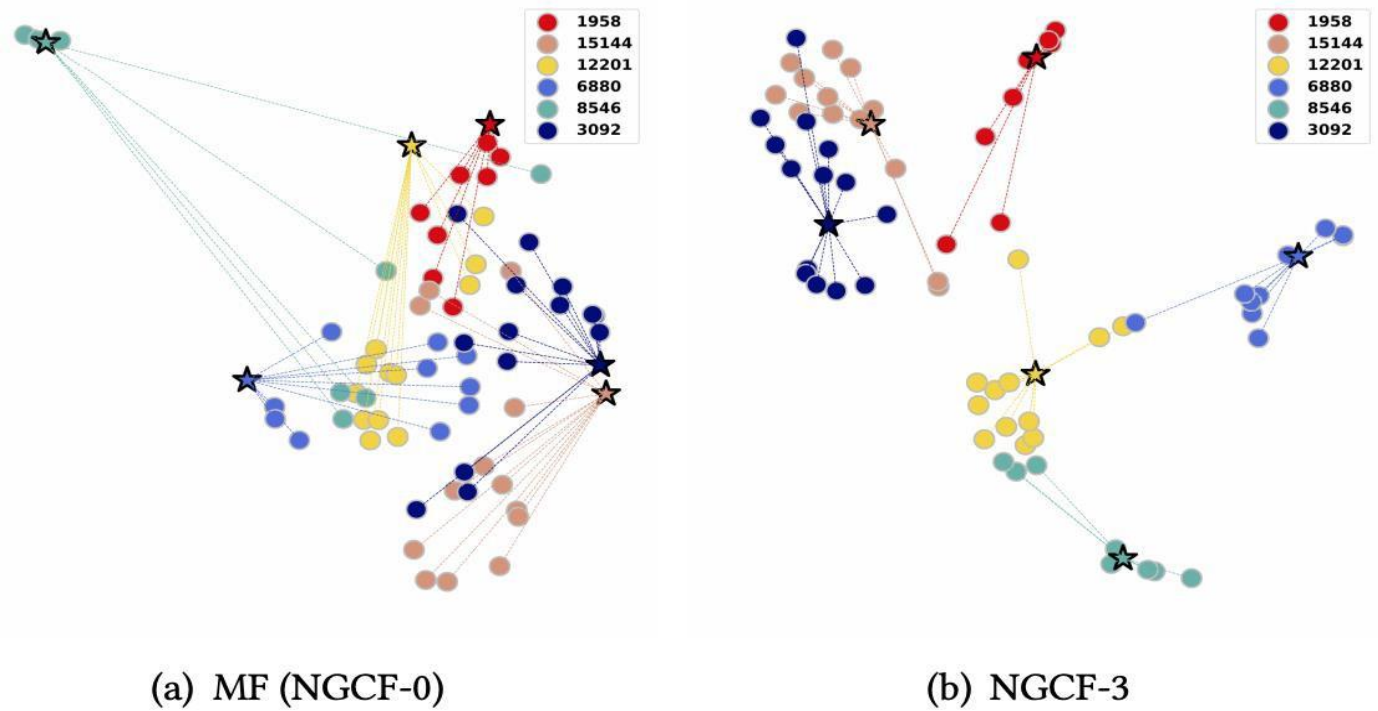




user groups with different group sparsity levels

- NGCF and HOP-Rec consistently outperform all other baselines on most user groups.
- Exploiting high-order connectivity facilitates the representation learning for inactive users.

Effect of High-order Connectivity



(a) MF (NGCF-0) (b) NGCF-3

Visualization of the learned t-SNE transformed representations. Star: user and Point: item

- The points with the same colors (i.e., the items consumed by the same users) tend to form the clusters.

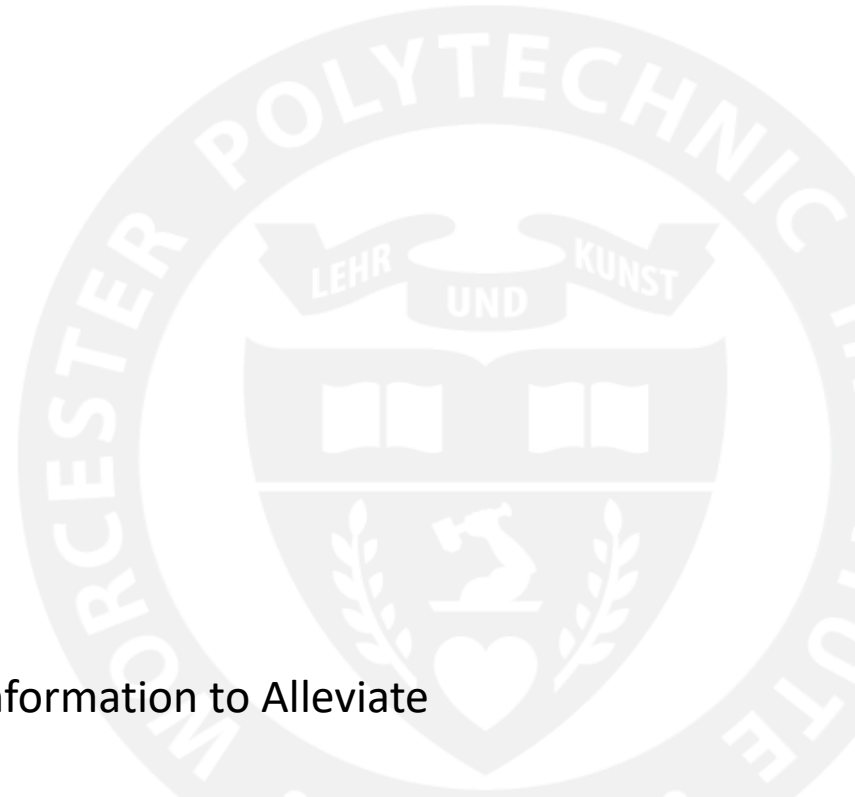
- The **connectivities of users and items are well reflected in the embedding space**, that is, they are embedded into the near part of the space.

Conclusion

- Take-home messages
  - Modeling high-order connectivity from user-item interactions is important for CF.
  - We proposed a GNN model to do this in an end-to-end way.

Where Are the Facts? Searching for Fact-checked Information to Alleviate the Spread of Fake News

EMNLP 2020



N. Vo, and K. Lee. Where are the Facts? Searching for Fact-checked Information to Alleviate the Spread of Fake News. EMNLP 2020.

What is fake news?

Fake news can be viewed as **wrong information** which is **deliberately created** to deceive people.

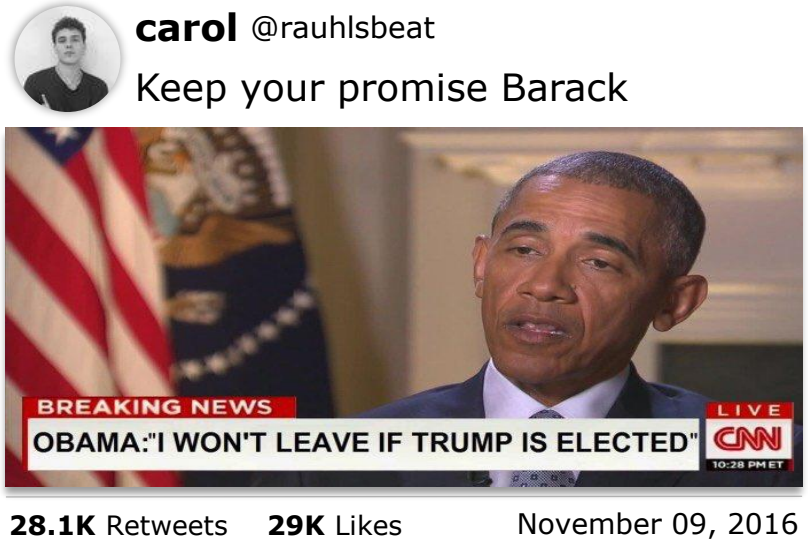
Russian propaganda effort helpe spread 'fake news' during election, experts say

Approaches to solve fake news

- Building ML models to detect fake news
- Fact-checking websites

Approaches to solve fake news

Despite the existence of these methods, fake news is still wildly spread.



The fake quote: *I won't leave if Trump is elected* is fact-checked by Snopes.com on September 2016

Two months later, 28.1K people still retweeted it.

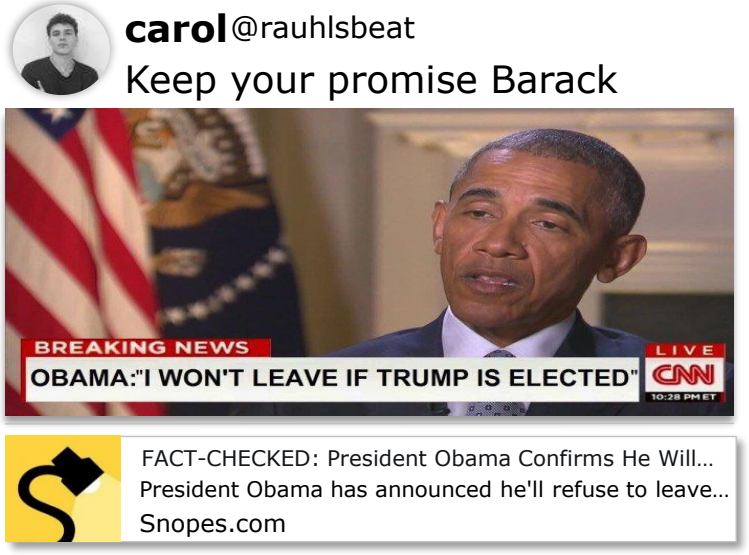
How can we discourage people from sharing fake news?

Our approach

Discouraging online users from sharing fake news

Incorporating fact-checked information into social media posts

Design



Mock-up

Benefits

- **Warning users about fake news**
- **Stopping the spread misinformation**
- **Scaling up volume of verified content**

Real life result

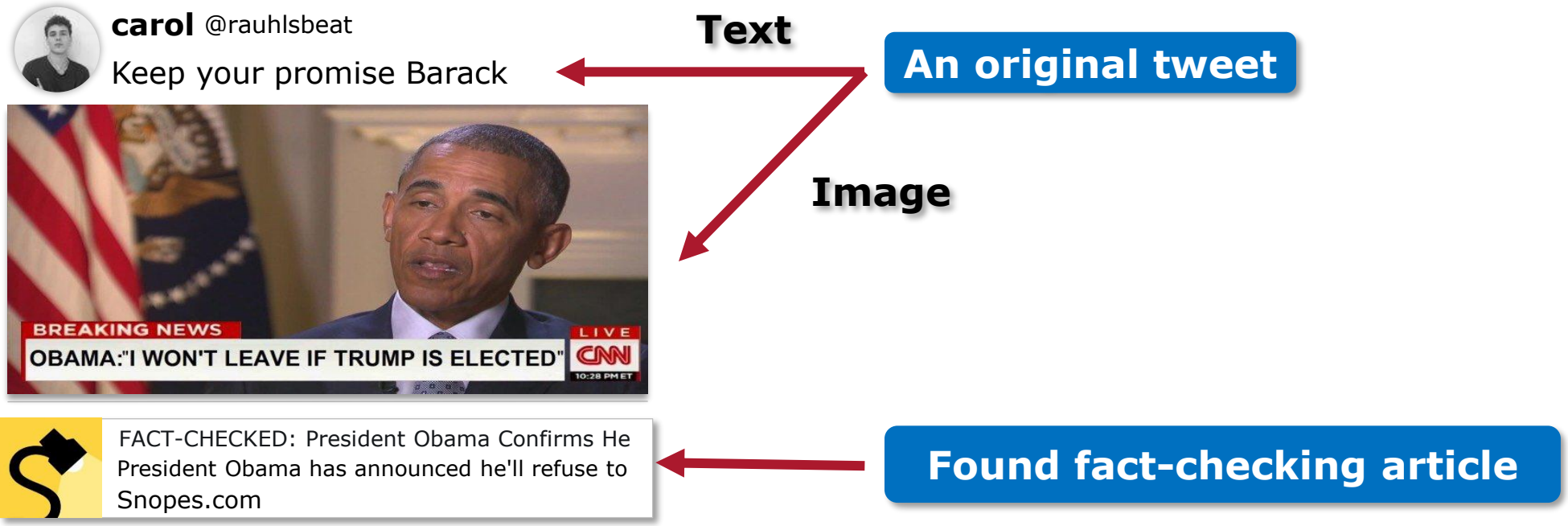
Over 95% of users did not further consume COVID-19 fake news when seeing fact-checking articles [1]

[1] How Facebook is combating spread of COVID-19 misinformation https://cnn.it/3gjtBkg

Goal: Incorporating Fact-checked Information Into Social Media Posts

How can we achieve the goal?

Searching for relevant fact-checking articles given multimodal content of social media posts



Our framework

Problem: Searching for fact-checking articles (documents) using text + images of original tweets (queries)

Step 1: Using BM25 to retrieve initial candidate documents

Step 2: Using a neural model for ranking refinement

A multimodal query

Keep your promise Barack

Document corpus

BM25

Initial candidates

Multimodal Attention Network

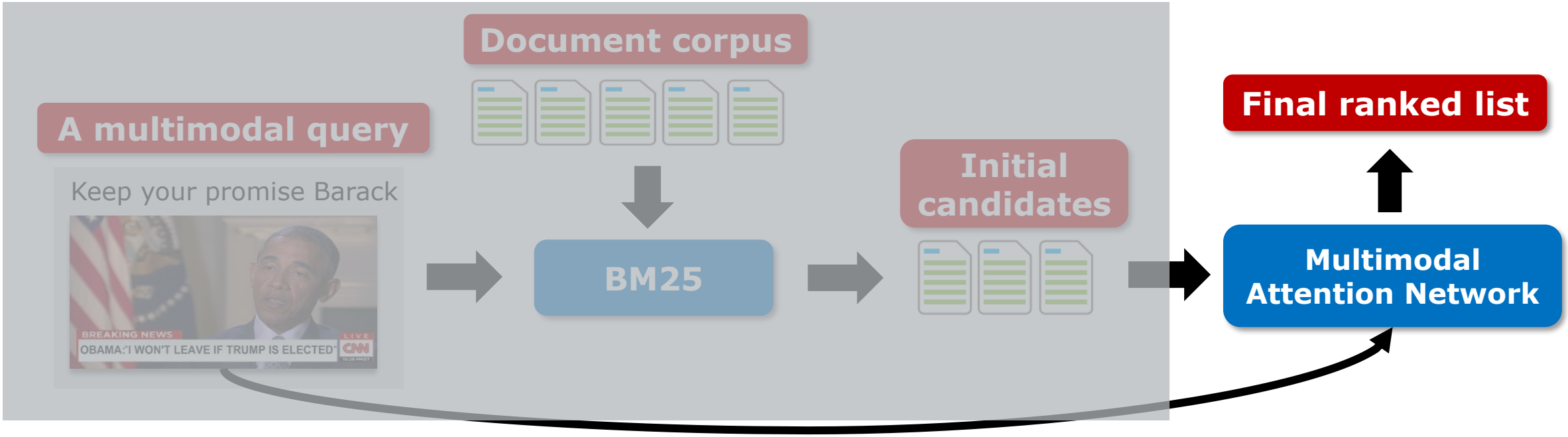
Final ranked list

Our framework

Problem: Searching for fact-checking articles (documents) using text + images of original tweets (queries)

Step 1: Using BM25 to retrieve initial candidate documents

Step 2: Using a neural model for ranking refinement

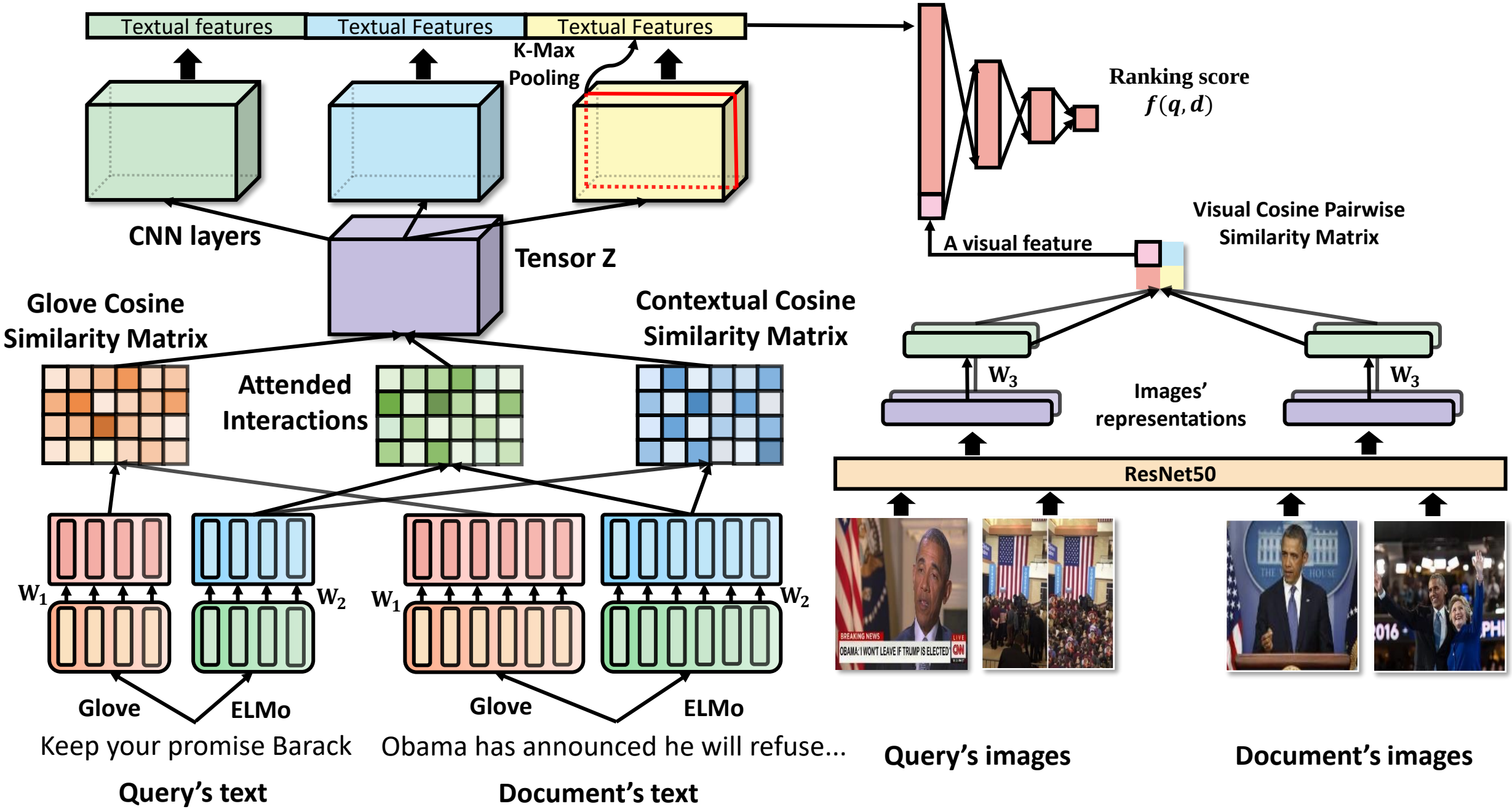


Multimodal Attention Network (MAN)

- Given an original tweet/query q and a document/FC-article d:
  - Compute $f(q, d)$ which represents relevancy between document d and query q
  - The **higher** $f(q, d)$ is, the **greater** chances that d fact-checks q

- To compute $f(q, d)$, our MAN uses:
  - textual matching signals
  - visual matching signals

Multimodal Attention Network (MAN)



Multimodal Attention Network

- We train our model with hinge loss:

$$\mathcal{L}(\theta) = \sum_{q} \sum_{d^+, d^- \in D_q} \max(0, 1 - f(q, d^+) + f(q, d^-))$$

θ is trainable parameters

Query q

$f(q, d^+)$ is score for a relevant article

$f(q, d^-)$ is score for a non-relevant article

Datasets

- We use two datasets
  - Snopes and PolitiFact
  - They are split into train/dev/test with ratio 80%/10%/10%

Datasets	Snopes			PolitiFact		
Items	Train	Dev	Test	Train	Dev	Test
|Tweets|	8002	1000	1001	1496	187	187
|Articles|	1703	1697	1697	467	467	467

Experiments

- We compare MAN with 10 baselines in 4 categories:
  - Keyword matching (1 model)
  - Multimodal retrieval methods (2 models)
  - Semantic matching methods (2 models)
  - Relevance matching methods (5 models)
- We test all models in two cases:
  - **Case 1:** Using (1) images and (2) text in tweets
  - **Case 2:** Using (1) images, (2) text in tweets and (3) text in images

Experiments

- We compare MAN with 10 baselines in 4 categories:
  - Keyword matching (1 model)
  - Multimodal retrieval methods (2 models)
  - Semantic matching methods (2 models)
  - Relevance matching methods (5 models)
- We test all models in two cases:
  - **Case 1:** Using (1) images and (2) text in tweets
  - **Case 2:** Using (1) images, (2) text in tweets and (3) text in images

Experiments

- Ranking experiments using (1) images, (2) text in tweets and (3) text in images.

		Snopes			PolitiFact		
Methods	Models	**NDCG@1**	**NDCG@3**	**NDCG@5**	**NDCG@1**	**NDCG@3**	**NDCG@5**
Multimodal retrieval	DVSHB	0.32667	0.46849	0.49640	0.21925	0.29335	0.32626
	TranSearch	0.45854	0.58410	0.61832	0.39572	0.50878	0.52397
Semantic Matching	ESIM	0.61139	0.70660	0.72999	0.33155	0.44658	0.48617
	NSMN	0.78821	0.85732	0.87148	0.58824	0.70002	0.73500
Relevance Matching	ConvKNRM	0.85914	0.90829	0.91401	0.66310	0.79163	0.80705
	CoPACRR	0.86913	0.91166	0.91851	0.66845	0.77419	0.79191
Ours	MAN	**0.88911**	**0.93343**	**0.93975**	**0.74332**	**0.84905**	**0.85987**
	Imprv. %	2.30%	2.39%	2.31%	11.20%	7.25%	6.54%

Our MAN is the best

Conclusions

- We propose a novel method to reduce the spread of misinformation
  - Searching and incorporating fact-checked information into social media posts
- We propose a novel neural ranking model
  - Using text + images of tweets to search fact-checked information
- Code and datasets:
  - https://github.com/nguyenvo09/EMNLP2020
- Contact: nkvo@wpi.edu

Acknowledgement: This work was supported in part by NSF grant CNS-1755536, AWS Cloud Credits for Research, and Google Cloud

Final Exam

- The exam will be held at 6pm next Wednesday in class.
- The exam is closed book.
- You may prepare and use one standard 8.5" by 11" piece of paper with any notes you think appropriate or significant.
- You may use a calculator if it make you feel comfortable. But no other electronic devices are allowed (e.g., cell phone, tablet and computer).

Project Website (8%)

- Your team should create a public website that shows off your project. You must use a well-known hosting service -- e.g., Google Sites, Github Pages -- so that we have some assurances your site will not disappear. We would expect to see the big questions and takeaways of your project. You should also include:
  - Link to your github repository containing your code and a brief README file to tell us how to set up our environment to run your code (e.g., are there dependencies we should be aware of?).
  - The data you used (raw or possibly a link to a cloud storage provider).
- Due Date: April 25 by 11:59pm

Project Presentation and In-class Q&A (11%)

- Final project presentation
  - Each team will have MAX 15 minutes for presentation including showing demo (if you have).
  - Submit your slides by April 25 11:59pm.
  - We will hold Q&A after your presentation.

The order of presentation on April 26

1. Bo Yu, Jin Yang, Kangjian Wu
2. Vignesh Sundaram, Amey More, Akanksha Pawar, Padmesh Naik
3. Xiaofan Zhou, Yiming Liu, Yuyuan Liu, Akhil Daphara
4. Sidhant jain, Parth dhruv, Darshan Swami, Aditya Ramesh
5. Chandra Rachabathuni, Ayush Shinde, Chao Wang, Anthony Chen, Adhiraj Budukh
6. Joe Scheufele, Alex Alvarez, Matt Suyer, Jason Dykstra
7. Adityavikram Gurao, Snehith Varma Datla, Sree Likhith Dasari, Supreeth Bandi
8. Samar, Bao, Michael, Megan