

---

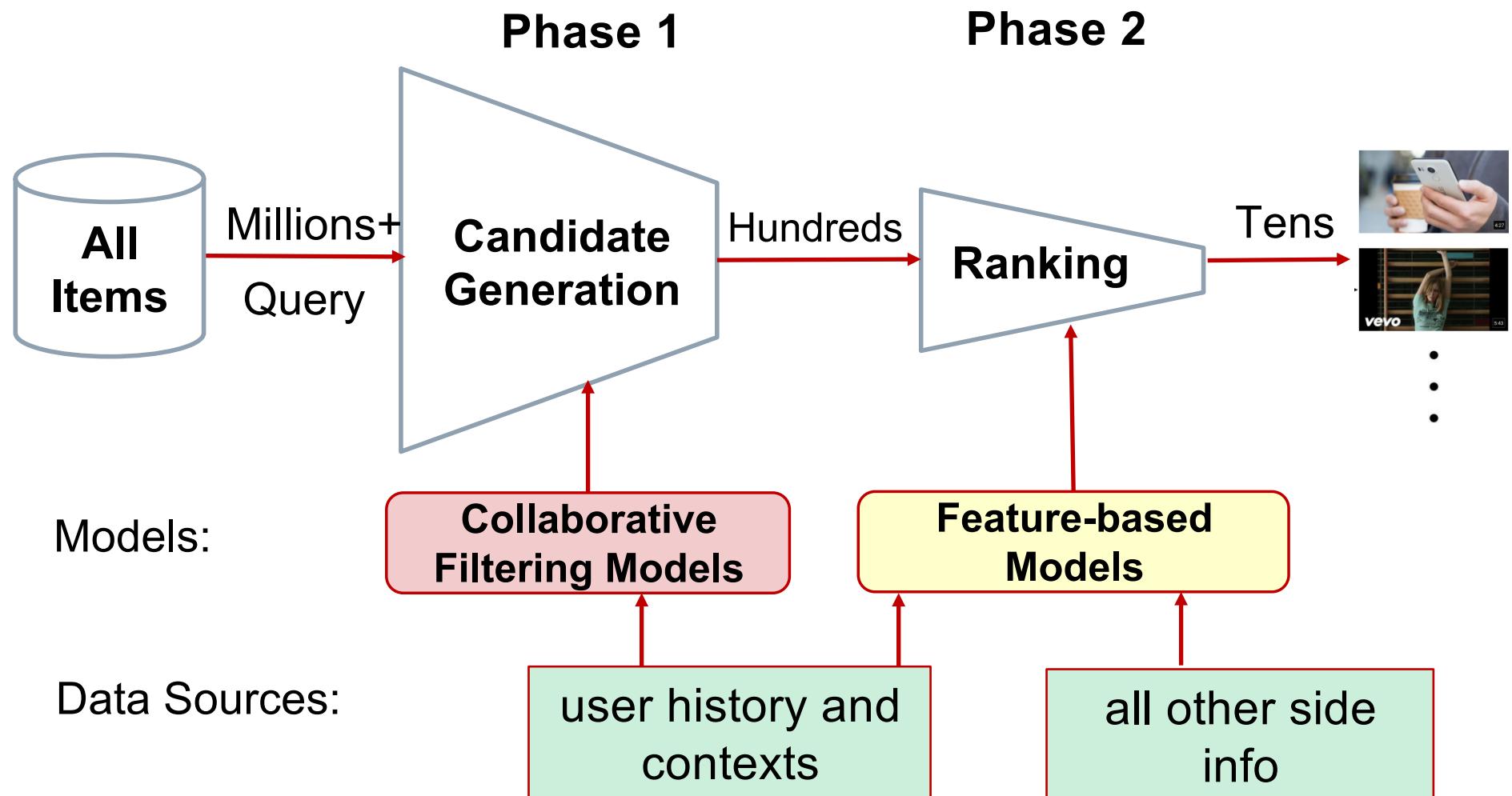
# **Recommendation in E-Commerce**

Presented by Xiangnan He

## Outline

---

- Overview of recommender systems (5 min)
- Candidate generation (35 min)
- Ranking methods (25 min)
- Summary (5 min)



## Need for candidate selection?

- Ranking is an expensive operation
  - Candidate selection is a first-pass filter to reduce input space to final ranking function
- Recommendations are highly personalized
  - Users expect a fast, updated, and contextually aware system
  - Online learning on selected candidate only
- Recall is more important than precision!
  - Generate a diverse set of items relevant to the query (e.g., user profile, target item, contexts)

- Efficiency is the key challenge:
  - Need to select candidates in **milliseconds**
  - Even scanning all items is infeasible --  $O(N)$  cost
  - The complexity needs to be **sublinear** in # of items.
- Two types of methods:
  1. Heuristic-based methods  
Define heuristics, e.g., co-occurrence, random walk etc.
  2. Embedding-based methods  
Learn embedding for user and item, and perform kNN search in the embedding space.

- Heuristic-based methods are usually simple and easy to implement
  - No objective function is optimized
- Based on item properties, e.g.:
  - Hot sale items
  - Promotion items
- Based on item relations, e.g.:
  - Similar items (metadata, co-purchase, visual, etc)
  - Random walk on item graph
  - Complementary and substitutable items

## Find similar items (item co-occurrence)

- Item co-occurrence statistics can be evaluated at different levels:
  - A user's whole history (long-term)
  - A session (short-term)
- *Relatedness* score between two items:

$$r(v_i, v_j) = \frac{c_{ij}}{f(v_i, v_j)}$$

Co-occurrence count  
Normalization function, e.g.,

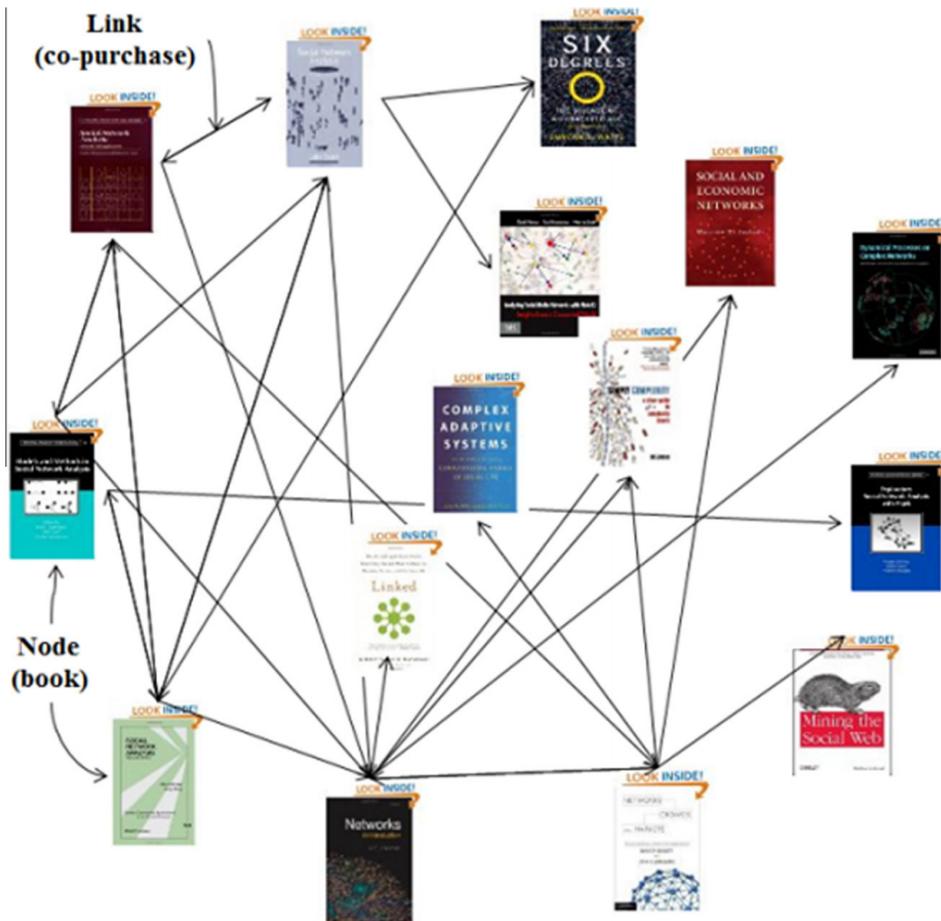
$$f(v_i, v_j) = c_i \cdot c_j$$

Product of items' global popularity

- Other normalization functions can be used, e.g., to define *transition probability*.

## Find similar items (item graph)

- Build an item graph based on co-occurrence:
  - Select a threshold to control graph density.



A typical workflow:

Step 1: Build item graph

Step 2: Define the target user's activity (e.g., purchased/clicked products) as **seeds**

Step 3: Propagating from the seeds on the graph to find more relevant items.

## Find similar items (graph random walk)

- Global propagation on graph is too costly:
  - Need to consider the whole graph structure and wait for convergence (consider PageRank)
- More efficient way is to do **local random walk!**
- Pinterest's Pixie system:
  1. Starting from each seed
  2. Performing many random walks (parallelized)
  3. Aggregate visit counts of covered items
- Need many tricks to ensure walking depth and personalization quality.
  - E.g., graph pruning and biased sampling

- Complementary and substitutable items:
  - Substitutes: items that are interchangeable (Co-view)



- Complements: items that might be purchased together (co-purchase)



- Substitutes and Complements are unique source to support specific scenarios:
  - Substitutes: “also viewed”, “buy after viewing”
  - Complements: “also bought”, “frequently buy together”
- Using candidates of substitutes and complements is beneficial to
  - Higher click-through rate
  - Higher conversation rate
  - Increase user stickiness

- Co-view and Co-purchase statistics provide weak labels for substitutes and complements.
  - Large statistics are more trustable
- How to learn from statistics and generalize to more products?
  - I.e., finding more item pairs having a relation.
- Can be formulated as a Link Prediction task with two types of relations.
  - Next: Wang et al. WSDM 2018. *A Path-constrained Framework for Discriminating Substitutable and Complementary Products in E-commerce*

## Basic model

- Input: item pairs that have a relation (either complementary or substitutable)
- Model: project items to embedding space
  - Basic Model:

$$P(y_{i,j}|V, V') = \sigma(v_i^T \cdot v_j'),$$

Probability that  $i$  has  
a relation with  $j$

Context vector of item  $j$

Target vector of item  $i$

- Parameters are learned by maximum likelihood:

$$\begin{aligned} L(Y|V, V') &= -\log(P(Y|V, V')) \\ &= - \sum_{(i,j) \in \mathcal{E}_P} \log(\sigma(v_i^T \cdot v_j')) \quad \text{Positive examples} \\ &\quad - N \cdot E_{j' \sim P_n(w)} [\log(\sigma(-v_i^T \cdot v_{j'}'))], \quad \text{Sampled negative examples} \end{aligned}$$

## Basic model + relation type + constraints

- Basic Model + Relation Type:
  - Project items from general semantic space to specific relation space

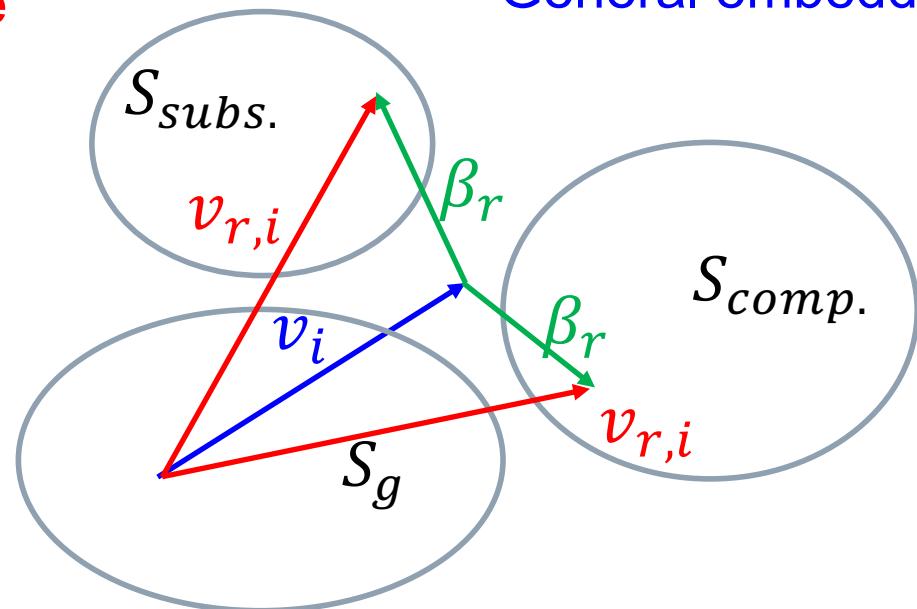
$$P(z_{i,j,r}) = \sigma(v_{r,i}^T \cdot v'_{r,j}),$$

Item vector in  
relation space

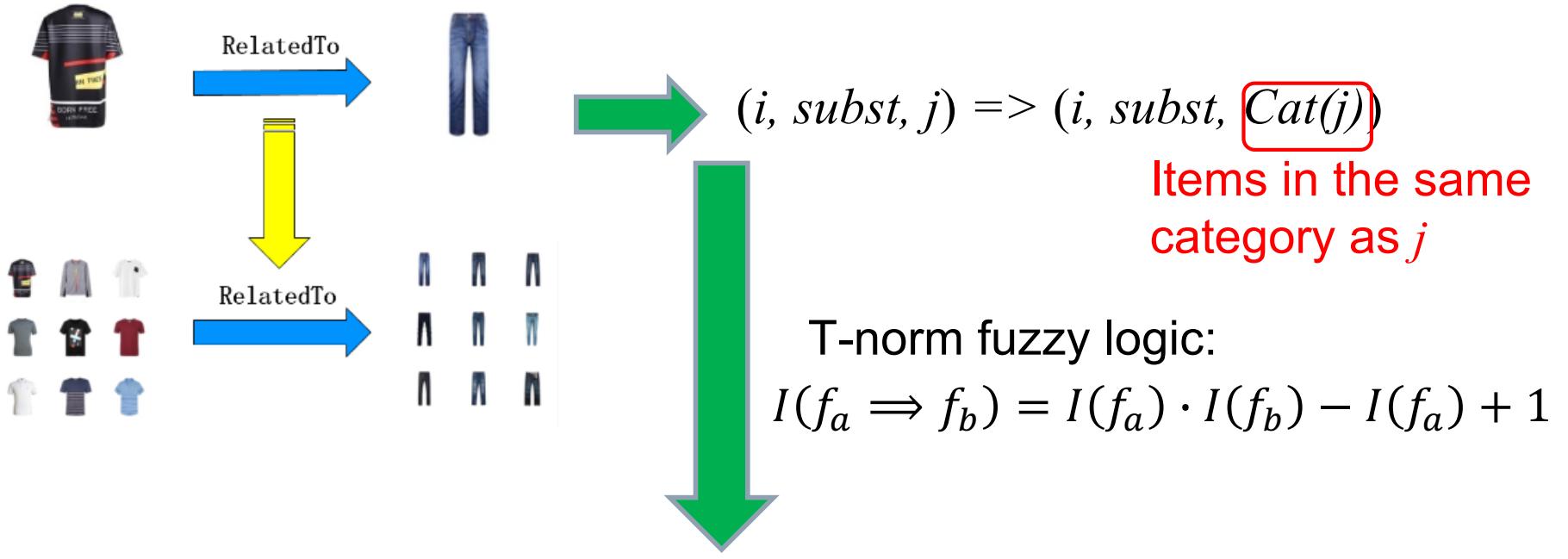
$$\begin{aligned} v_{r,i} &= v_i + \beta_r \odot v_i \\ v'_{r,i} &= v'_i + \beta_r \odot v'_i, \end{aligned}$$

Relation type vector

General embedding



- Constraints can be enforced to alleviate label sparsity.
  - Constraint 1: Product category constraint.



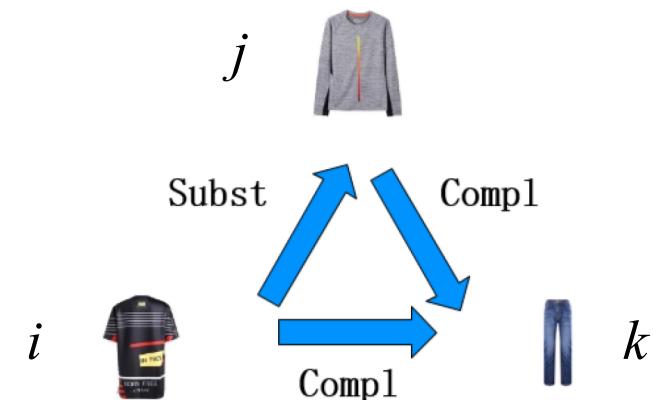
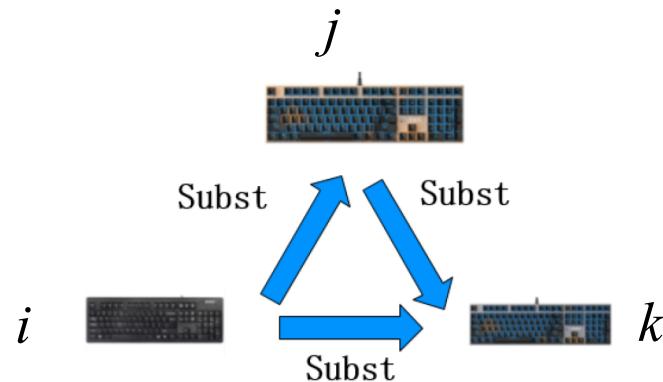
If  $(i, r, j)$  &&  $k$  is in  $\text{Cat}(j)$

Then the probability that  $(i, r, k)$  is a **positive example** is:

$$I(f_a \Rightarrow f_b) = P(z_{i,j,r})P(z_{i,k,r}) - P(z_{i,j,r}) + 1$$

## Basic model + relation type + constraints

- Constraints can be enforced to alleviate label sparsity.
  - Constraint 2: Multi-Step Path Constraints.



$$(i, \text{subst}, j) \wedge (j, \text{subst}, k) \Rightarrow (i, \text{subst}, k) \quad (i, \text{subst}, j) \wedge (j, \text{compl}, k) \Rightarrow (i, \text{compl}, k)$$

Similarly, there are two more such constraints:

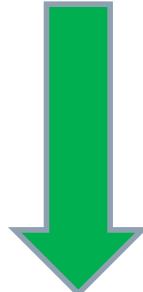
$$(i, \text{compl}, j) \wedge (j, \text{compl}, k) \Rightarrow (i, \text{compl}, k) \quad (i, \text{compl}, j) \wedge (j, \text{subst}, k) \Rightarrow (i, \text{compl}, k)$$

- Constraints can be enforced to alleviate label sparsity.
  - Constraint 2: Multi-Step Path Constraints.

Generic form

$$f_2 \triangleq (i, r_1, j) \wedge (j, r_2, k) \Rightarrow (i, r_3, k)$$

$(i, subst, j) \wedge (j, subst, k) \Rightarrow (i, subst, k)$   
 $(i, subst, j) \wedge (j, compl, k) \Rightarrow (i, compl, k)$   
 $(i, compl, j) \wedge (j, compl, k) \Rightarrow (i, compl, k)$   
 $(i, compl, j) \wedge (j, subst, k) \Rightarrow (i, compl, k)$


**T-norm fuzzy logic:**  
 $I(f_a \wedge f_b) = I(f_a) \cdot I(f_b)$   
 $I(f_a \Rightarrow f_b) = I(f_a) \cdot I(f_b) - I(f_a) + 1$

If  $(i, r_1, j)$  &&  $(j, r_2, k)$   
 Then the probability that  $(i, r_3, k)$  is a **positive example** is:

$$I(f_2) = P(z_{i,j,r1}) \cdot P(z_{j,k,r2}) \cdot P(z_{i,k,r3}) - P(z_{i,k,r1}) \cdot P(z_{j,k,r2}) + 1$$

## Basic model + relation type + constraints

- Joint learning over the basic model, relation type, and all constraints:

Relation type vector:  $v_{r,i} = v_i + \beta_r \odot v_i$

$$\underline{L(Y|V,V')} + \alpha_0 \cdot L(Z|V,V',\beta) + \alpha_1 \cdot L(F_1|V,V',\beta) + \alpha_2 \cdot L(F_2|V,V',\beta)$$

Basic model

Basic model  
+ relation type

Basic model  
+ relation type  
+ category  
constraint

Basic model  
+ relation type  
+ category  
constraint  
+ path constraints

- ✓ Heuristic methods:
  - ✓ How to find similar items
  - ✓ How to find complementary and substitutable items

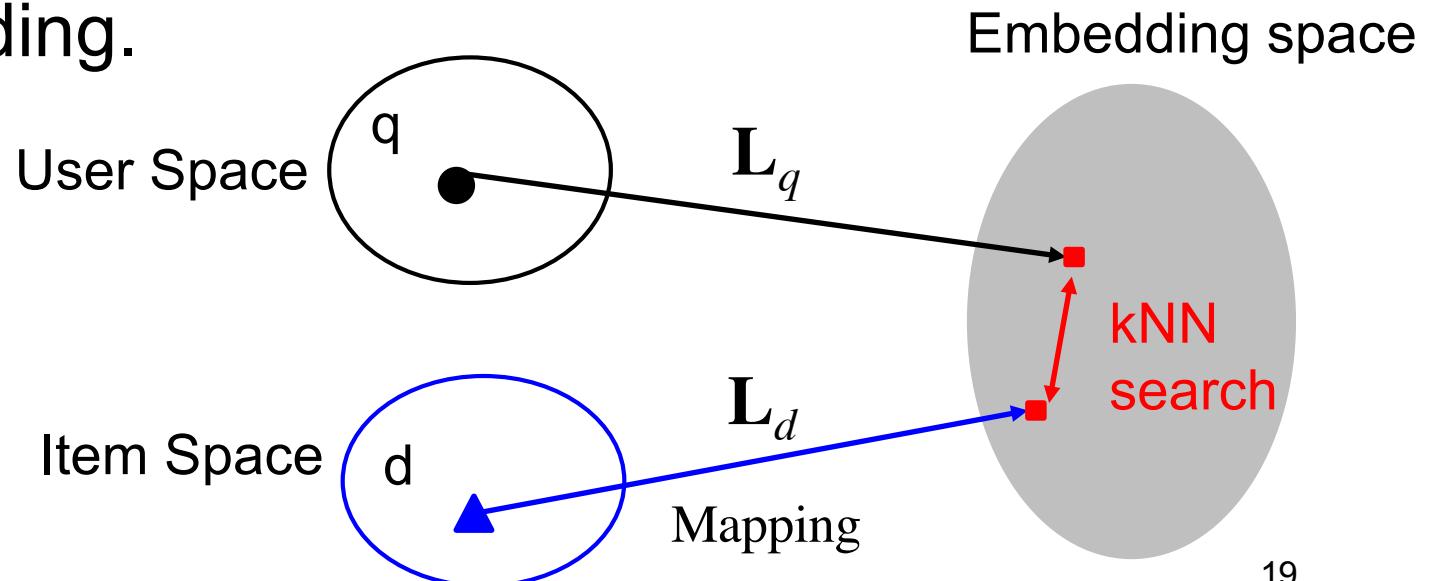
✓ Heuristic methods:

- ✓ How to find similar items
- ✓ How to find complementary and substitutable items

In the next...

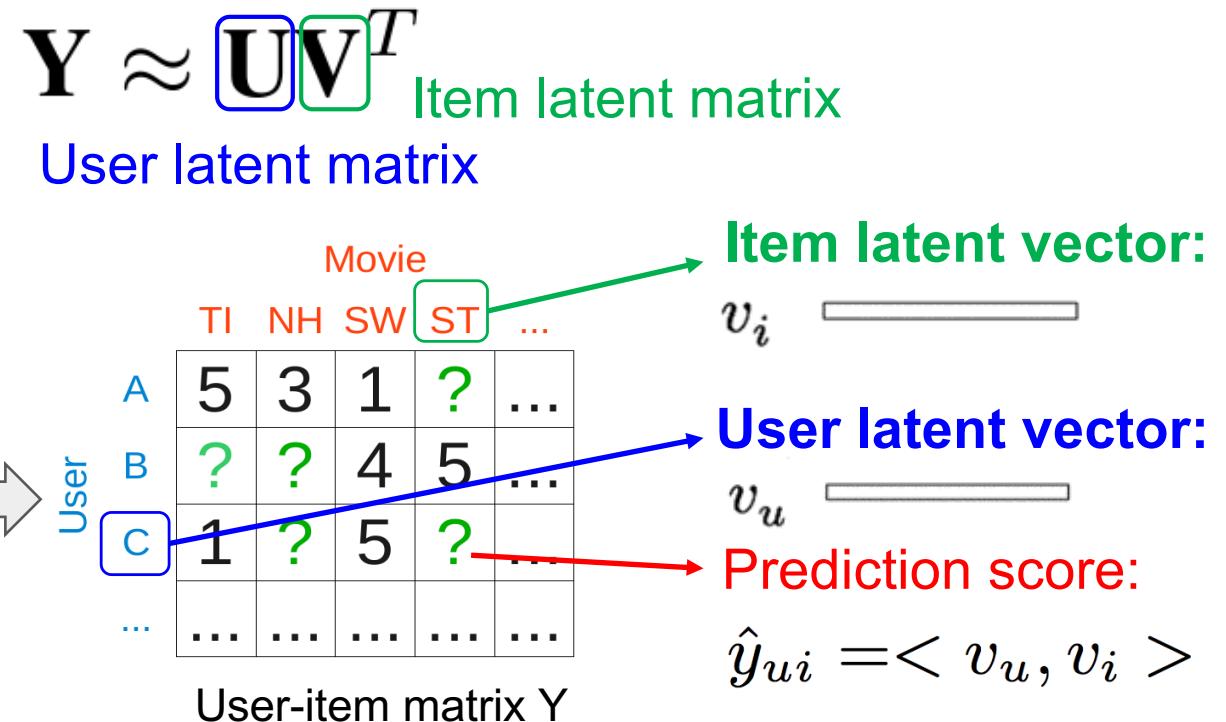
- Embedding-based methods

- The key is to learn user embedding and item embedding.



## Matrix factorization model

- Matrix Factorization (MF) assumes the user-item interaction matrix has a low-rank structure:



- Each user and item is described as an embedding vector
- The score is estimated as the **inner product** of user embedding and item embedding

## User embedding + rating history

- MF profiles a user with an ID, directly projecting ID to embedding space.
  - Taking out the  $u$ -th row in user embedding matrix  $\mathbf{U}$
- Another more **information-rich way** for user profile is to use the **rating history**:

$$\hat{y}_{ui} = \left( \frac{1}{|\mathcal{R}_u|^\alpha} \sum_{j \in \mathcal{R}_u} \mathbf{q}_j \right)^T \mathbf{v}_i$$

$\Leftrightarrow$  user embedding

Can be interpreted as the similarity between item  $i$  and  $j$

Items rated by  $u$

- Known as *factored item similarity model* (FISM, Kabbur et al, KDD'14), since it factorizes item similarity matrix into two low-rank matrices.

## User embedding + rating history

- MF profiles a user with her ID
  - ID embedding encodes user general interest
- FISM profiles a user with her interacted items
  - Recommend items that are similar to historical items.
- We can profile a user with both her ID and rating history:

$$\hat{y}_{ui} = \underbrace{(\mathbf{v}_u + \frac{1}{|\mathcal{R}_u|^\alpha} \sum_{j \in \mathcal{R}_u} \mathbf{q}_j)^T \mathbf{v}_i}_{\text{User ID Embedding}} + \underbrace{\mathbf{v}_u^T \mathbf{v}_i}_{\text{Rating History Embedding}}$$

- Known as SVD++ model (Koren, KDD'08), the best single model for rating prediction in Netflix challenge (3 years, 1 million price)

## User embedding + contexts

- User purchase decisions are context-aware:
  - Contexts: time, location, query, weather etc.
- Context-aware MF model:

$$\hat{y}_{ui} = (\mathbf{v}_u + \sum_{c \in C(u)} \mathbf{v}_{u,c})^T \mathbf{v}_i$$

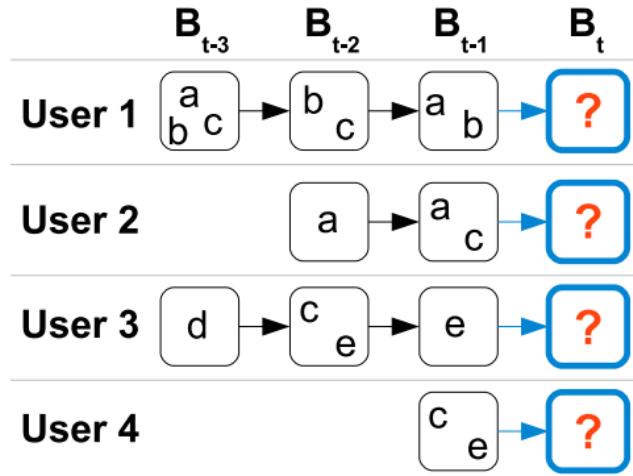
User Embedding  
Context-aware User Embedding  
Current contexts

- When a context is dense, we can directly learn  $\mathbf{v}_{u,c}$  from data, e.g., timeSVD model (Koren et al, KDD'09)
- When the context is sparse, we can decompose  $\mathbf{v}_{u,c}$  to reduce model parameters to avoid overfitting, e.g.:

$$\mathbf{v}_{u,c} = \mathbf{v}_c$$
$$\mathbf{v}_{u,c} = \mathbf{v}_u \odot \mathbf{v}_c$$

## User embedding + recent purchases

- Next-basket recommendation in E-commerce:



Two properties:

1. User purchases a **basket** of products at a time
2. User behaviors are sequential: which products will be purchased in **next basket**?

- Factorizing Personalized Markov Chain (FPMC, Rendle et al, WWW'10):

$$\hat{y}_{uit} = (\mathbf{v}_u + \sum_{l \in B_{t-1}} \mathbf{v}_l)^T \mathbf{v}_i$$

User Embedding      Estimate transition probability from last item  $l$  to next item  $j$   
Items purchased in last basket

## Item embedding + attributes

- Attributes of item side can be integrated by using similar way of embedding addition:
  - E.g., product tags, category, price

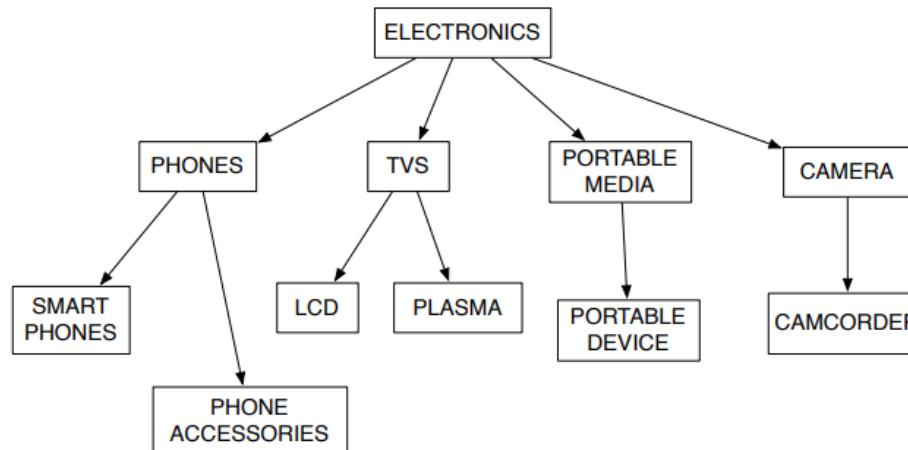
$$\hat{y}_{ui} = \mathbf{v}_u^T \left( \mathbf{v}_i + \sum_{a \in A(i)} \underline{\mathbf{v}_a} \right)$$

Item Embedding    Attribute Embedding

- In the next, we consider two special “attributes” of E-commerce products:
  - Taxonomy (i.e., category tree)
  - Product Image

## Item embedding + taxonomy

- Products are typically organized into hierarchical taxonomy, e.g., Electronics products:



- Taxonomy-aware MF (Kanagal et al, VLDB'12):

$$\hat{y}_{ui} = \mathbf{v}_u^T (\mathbf{v}_i + \sum_{l \in Path(i)} \mathbf{v}_l)$$

Item Embedding

Embedding of intermediate node  $l$  in the path

Path from item  $i$  (leaf node) to root node in the taxonomy

- Another way is to use taxonomy to **regularize the learning** of item embeddings.
  - Key assumption: Products of the same category should be **more close** with each other in the embedding space.
  - Hierarchical regularizer (Menon et al, KDD'11):

$$Reg(\mathbf{V}) = \sum_{i \in \mathcal{I} \cup \mathcal{T}} (\mathbf{v}_i - \mathbf{v}_{Parent(i)})^2$$

Parent of node  $i$  in the taxonomy  
All items and taxonomy nodes

- The regularizer can be added to any objective function --- a universal way to incorporate taxonomy in recsys.

Product images are particularly useful for some categories, such as fashion products.

What user purchased:



U.S. Polo Assn. Men's V-  
Neck T-Shirt  
★★★★★ 781  
\$9.25 - \$40.95



Nautica Men's Short Sleeve  
Crew Neck Cotton T-Shirt  
★★★★★ 12  
\$13.30 - \$39.10



Nautica Men's Solid Crew  
Neck Short Sleeve Pocket  
T-Shirt  
★★★★★ 34  
\$18.17 - \$65.57

Will user like this one?



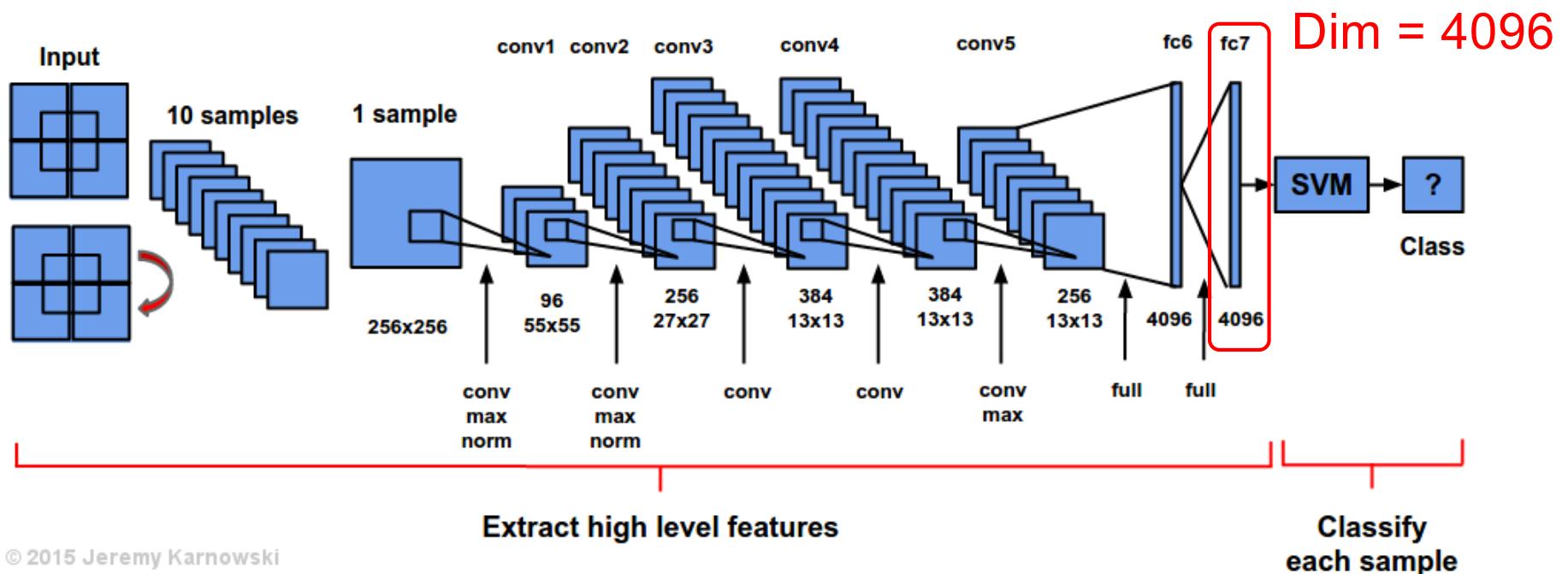
Each product = ID + image

Two key questions:

1. How to understand image?
2. How to integrate image feature into CF model?

## Image understanding

- Traditional (low-level) image features:
  - Pixels, Color histograms
  - SIFT descriptors
- Gap between low-level features and real semantics.
- Recent work uses deep CNN as feature extractor.



## Item embeddings + image

- Let  $\mathbf{f}_i$  be CNN features for image  $i$ :
  - Usually of thousands dimension. E.g., AlexNet: 4096, ResNet: 2048
- MF predicts user rating on image  $i$ :

$$\hat{y}_{ui} = \langle \mathbf{p}_u, \mathbf{f}_i \rangle = \mathbf{p}_u^T \mathbf{f}_i$$

User preference on image CNN features

- Problem:
  - $\mathbf{p}_u$  has to be of the **same dimension** as  $\mathbf{f}_i$
  - Too big latent space: **too many parameters => overfitting**
    - E.g., 100 million users \* 4096 \* 8 B = 3.28 TB
  - Typically, the dimension of CF latent space is hundreds (128, 256) at most.

- An intuitive solution is to do dimension reduction on CNN features, e.g., PCA
  - However, it will lose signal in CNN features.
  - The objective of dimension reduction is not recommendation.
- Solution: learning a **transformation matrix** to do the projection based on user-item interactions:

$$\hat{y}_{ui} = \mathbf{p}_u^T (\mathbf{E} \mathbf{f}_i)$$

Transformation matrix that projects  
CNN features to latent space

- $\mathbf{E}$  is optimized for recommendation task.
- This is the model widely used deep feature-based recsys (Geng et al, ICCV'15, He et al, AAAI'16).

## Item embedding + image

- User may care about different parts on a product:



(Chen et al, 2018)

- User-sensitive Image Representation

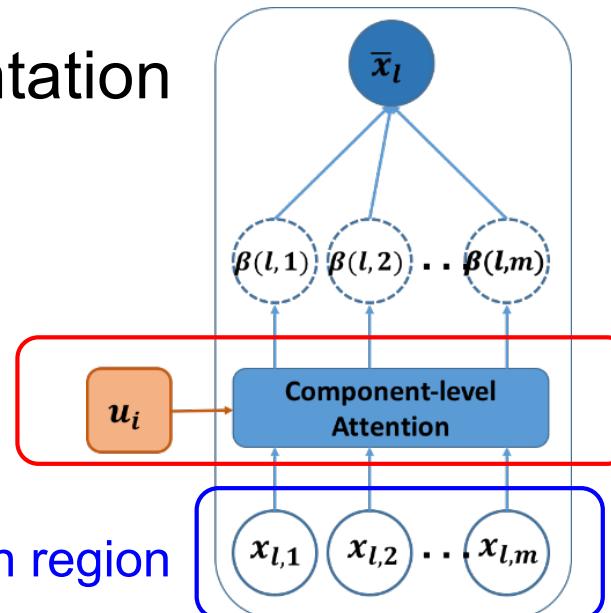
(Chen et al, SIGIR'17)

Attention Net determines region's weight:

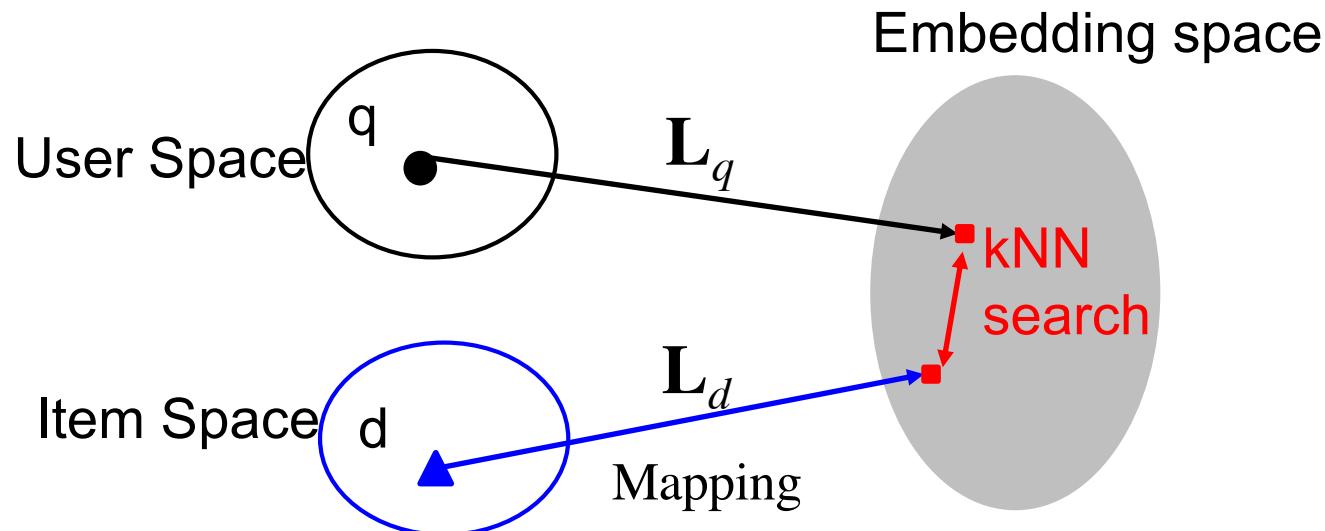
- Input: user embedding and region feature

$$b(i, l, m) = \mathbf{w}_2^T \phi(\mathbf{W}_{2u} \mathbf{u}_i + \mathbf{W}_{2x} \mathbf{x}_{lm} + \mathbf{b}_2) + \mathbf{c}_2$$

Feature of each region



## Short summary



- We have covered the model design for user embedding and item embedding.
  - User Embedding + Rating History / Contexts / Recent Purchases
  - Item Embedding + Attributes / Taxonomy / Image
- It is natural to combine all above info to build a unified embedding-based predictive model.
- **Next: how to learn embeddings?**
  - I.e., the optimization process

## User feedback data

### Explicit Feedback conveys user preference explicitly

- E.g., user ratings
- Usually real-values
- Higher score => positive signal
- Lower scores => negative signal

		Movie				
		TI	NH	SW	ST	...
User	A	5	3	1	?	...
	B	?	?	4	5	...
	C	1	?	5	?	...
	...	...	...	...	...	...

### Implicit Feedback conveys user preference implicitly:

- E.g., clicks, purchases
- Usually binary 0/1
- Observed data => positive signal
- Unobserved => negative signal

		Movie				
		TI	NH	SW	ST	...
User	A	1	1	1	?	...
	B	?	?	1	1	...
	C	1	?	1	?	...
	...	...	...	...	...	...

In E-Commerce, most feedback data are **implicit feedback**, much more than explicit ratings!

- Cheaper and easier to collect (e.g., server logs)

## Rating prediction is suboptimal

- Old-style work on recommendation optimize L2 loss on observed user-item interactions:

$$L = \sum_{(u,i) \in \mathcal{R}} w_{ui} (\hat{y}_{ui} - \hat{y}_{ui})^2 + \lambda \left( \sum_u \|\mathbf{v}_u\|^2 + \sum_i \|\mathbf{v}_i\|^2 \right)$$

Observed interactions

- But many empirical evidence show that:  
**A lower error rate does not lead to a good ranking performance...**
- Possible Reasons:
  - 1) **Discrepancy** between error measure (e.g., RMSE) and ranking measure.
  - 2) **Survival bias** – users tend to consume items they like and ignore items they dislike  
=> important to account for **missing data!**

## Towards top-K recommendation

- Recommendation is a personalized ranking task by nature, rather than rating prediction (regression).
  - Evaluated by Precision/Recall/AUC etc, rather than RMSE!
- Optimizing the **relative ranking** of a user on two items are more advantageous:
  - Higher rating > Lower rating (explicit feedback)
  - Observed interaction > Unobserved interaction (implicit feedback)

$$L_{BPR} = \arg \max_{\Theta} \sum_{(u, i, j) \in \mathcal{R}_B} \ln \sigma(\hat{y}_{ui} - \hat{y}_{uj}) - \lambda \|\Theta\|^2$$

sigmoid      Positive prediction      Negative prediction

Pairwise training examples:  $u$  prefers  $i$  over  $j$

- Known as the Bayesian Personalized Ranking objective (BPR, Rendle et al, UAI'09)

## Training procedure

$$L_{BPR} = \arg \max_{\Theta} \sum_{(u, i, j) \in \mathcal{R}_B} \ln \sigma(\hat{y}_{ui} - \hat{y}_{uj}) + \lambda \|\Theta\|^2$$

Pairwise examples:  $u$  prefers  $i$  over  $j$

For a positive instance  $(u, i)$ , e.g., a purchase, all non-purchased items of  $u$  can be used as negative instances.

$$\mathcal{D} := \{(u, i, j) | i \in \mathcal{Y}_u^+ \wedge j \notin \mathcal{Y}_u^+\}$$

Items purchased by  $u$     Items not purchased by  $u$

Using SGD (stochastic gradient descent) for optimization:

- Step 1: Sample a positive instance  $(u, i)$
- Step 2: Sample a negative instance  $(u, j)$  to pair with  $(u, i)$
- Step 3: Update parameters w.r.t. this stochastic instance

Which negative items to sample?

## Static negative sampling

- Sampling from a static distribution:
  - Uniform distribution: all items are equally likely to be disliked by user (vanilla BPR sampler)
  - Popularity-aware distribution: popular items are more likely to be disliked by user (He et al, SIGIR'16)
- Note that the objective of BPR learning is to increase the margin  $\hat{y}_{ui} - \hat{y}_{uj}$  as much as possible.
  - Problem: If the current model already scores  $(u, i)$  much higher than  $(u, j)$ , sampling  $(u, j)$  as negative **has fewer gain** to model update.
  - In other words, **the gradients of BPR objective w.r.t.  $(u, i, j)$  are close to 0.**

## Dynamic negative sampling

- Basic idea: sampling **hard negatives** leads to more gain to the current model.
  - The negative sampling distribution **dynamically changes** with model updates.
- Difficulty: evaluating model prediction on all items is too time-consuming!
- An approximate algorithm to DNS:
  - Step 1: Randomly sample  $K$  negatives (e.g.,  $K=20$ )
  - Step 2: Use current model to score the  $K$  negatives
  - Step 3: Pick the negative with the highest score as the “true negative” for model update.
- DNS always leads to **faster convergence** and **better performance**.

## Other optimization alternatives

- Besides optimizing BPR objective, there are other optimization choices, e.g.:

- Margin-based pairwise loss:

$$Loss = \sum_{(u,i,j) \in \mathcal{D}} \max(0, \Delta + \hat{y}_{uj} - \hat{y}_{ui})$$

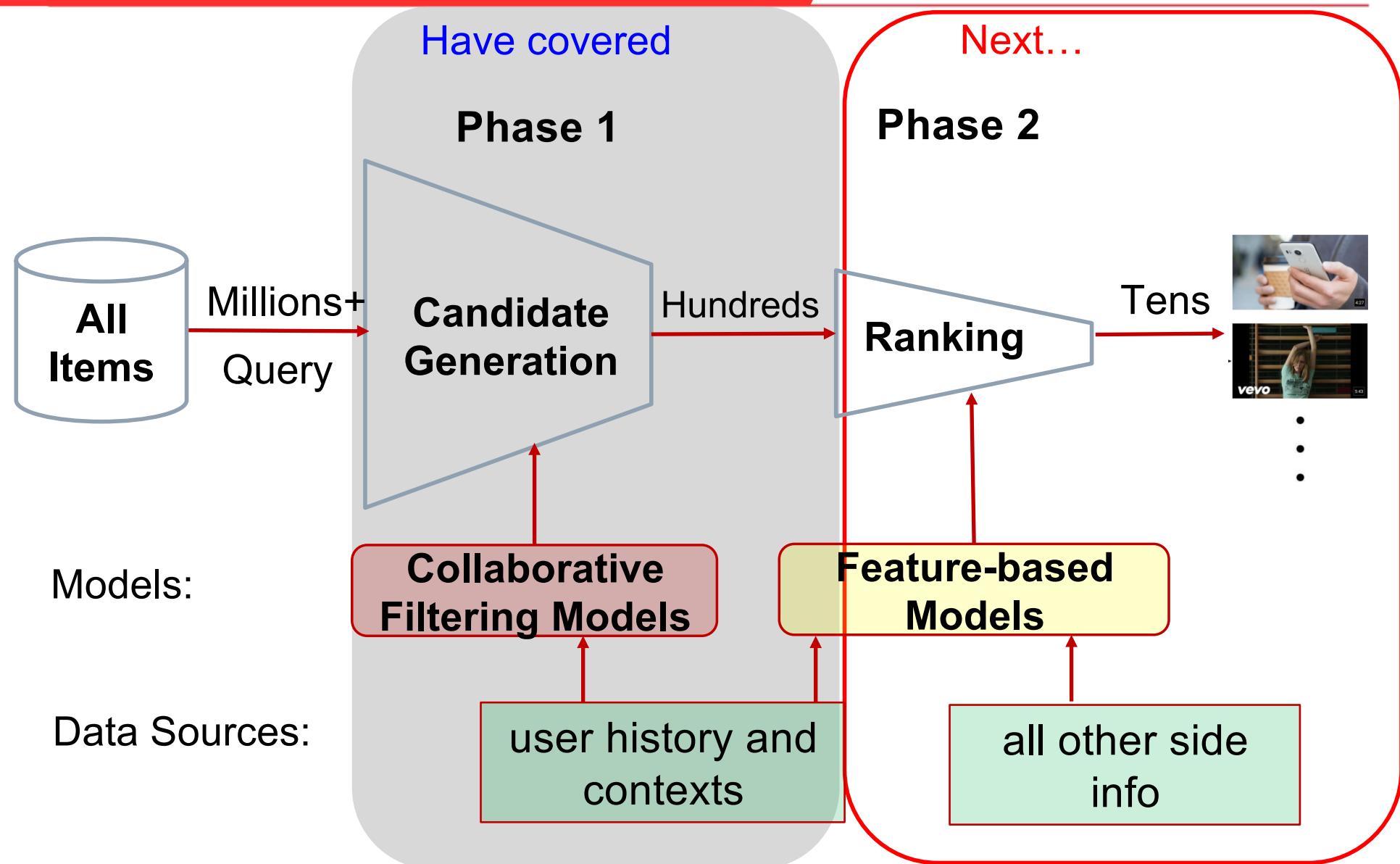
Expected minimum margin between positive prediction and negative prediction

- More commonly used in Knowledge Graph Completion
- Recently used in RecSys (Ying et al, KDD'18)
- Point-wise classification loss:

$$L = \sum_u \left[ \underbrace{\sum_{i \in \mathcal{R}_u} \log \sigma(\hat{y}_{ui})}_{\text{Observed interactions (positive)}} + w_0 \underbrace{\sum_{j \in \mathcal{R}_u^-} \log(1 - \sigma(\hat{y}_{uj}))}_{\text{Unobserved interactions (negative)}} \right]$$

- Treat recommendation as a classification task.
- Also commonly used in RecSys (He et al, WWW'17)

## Recommender system overview



## Requirements for ranking

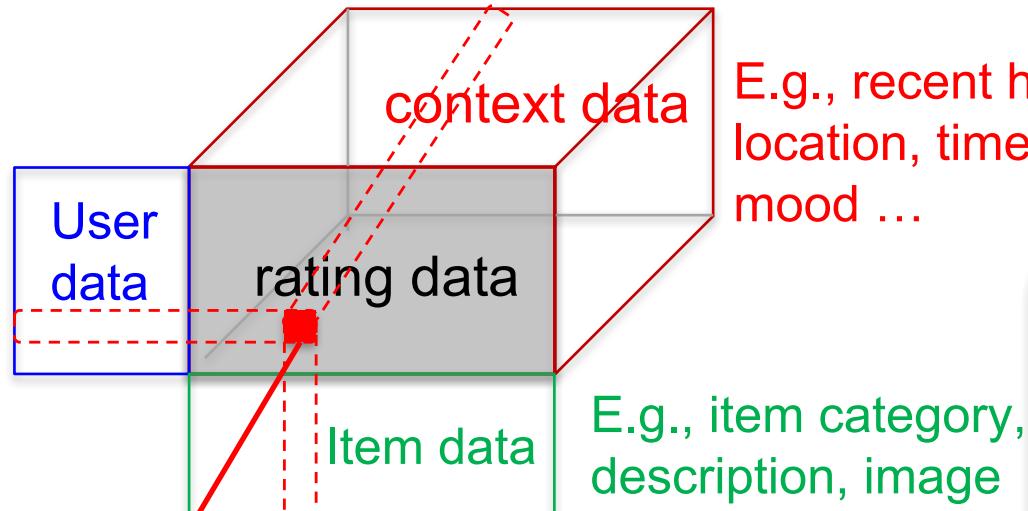
---

- Fine-tuning the results to present to end user
  - Ensemble different candidate sources
- Only a few hundred items are being scored
  - Low requirement on model efficiency
  - More complicated models can be used
  - More features can be used, e.g., user profiles, contexts, combinatorial features ...
- Optimization objective can be tailored for different scenarios, such as optimizing for:
  - Click-through rate (CTR)
  - Purchase
  - Impression

.....

## Input to feature-based models

E.g., user gender, age, occupation, personality ...



E.g., item category, description, image

E.g., recent history, location, time, weather, mood ...

Each row encodes all info for a rating

	Feature vector $\mathbf{x}$										
	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$x^{(4)}$	$x^{(5)}$	$x^{(6)}$	$x^{(7)}$	A	B	C	...
	1 0 0 ...	1 0 0 ...	0 0 1 ...	0 0 1 ...	0 0 0 1 ...	1 0 0 0 ...	0 0 1 0 ...	TI	NH	SW	ST ...
											Movie
											Other Movies rated ...

	Target $y$
5	$y^{(1)}$
3	$y^{(2)}$
1	$y^{(3)}$
4	$y^{(4)}$
5	$y^{(5)}$
1	$y^{(6)}$
5	$y^{(7)}$

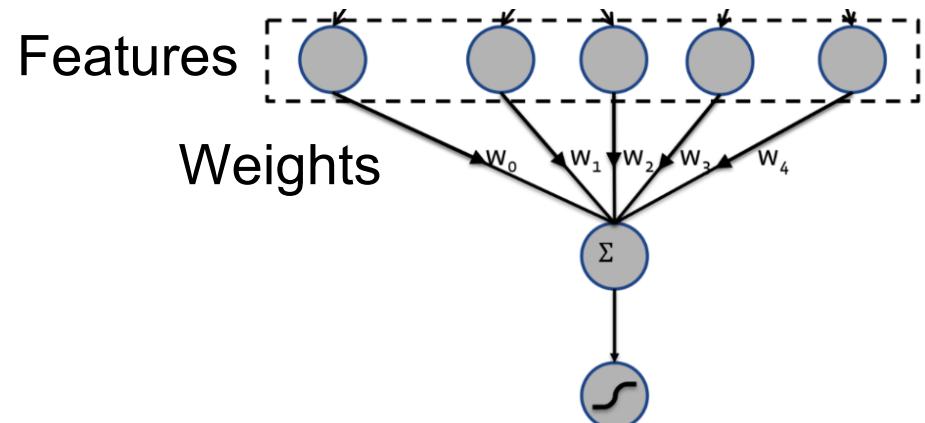
**Input Features:**

1. Categorical features: *user/item ID, bag-of-words, cross features...*
2. Numerical features: *textual/visual embeddings, converted features (e.g. TFIDF, GBDT)...*

Predictive Model

- Feature vector is high-dimensional but sparse
  - Consider the CF case: feature vector = user ID + item ID
  - Need to discover prediction patterns in nonzero features
- The interactions between features are important
  - E.g., users like to use **food delivery apps** at **meal-time**  
=> Order-2 interactions between app **category** and **time**
  - E.g., **male teenagers** like **shooting games**  
=> Order-3 interactions between **gender**, **age**, and **app category**.
- Crucial for feature-based models to capture **feature interactions** (aka., cross features)

## Logistic regression (LR)



Model Equation:

$$\hat{y}(\mathbf{x}) = \mathbf{w}^T \mathbf{x} = \sum_{i=1}^n w_i x_i$$

- Each feature  $i$  has a weight  $w_i$

An example of CTR prediction:

Publisher	Advertiser
ESPN	Nike

$$\rightarrow s = w_{\text{ESPN}} + w_{\text{Nike}}$$

Pros:

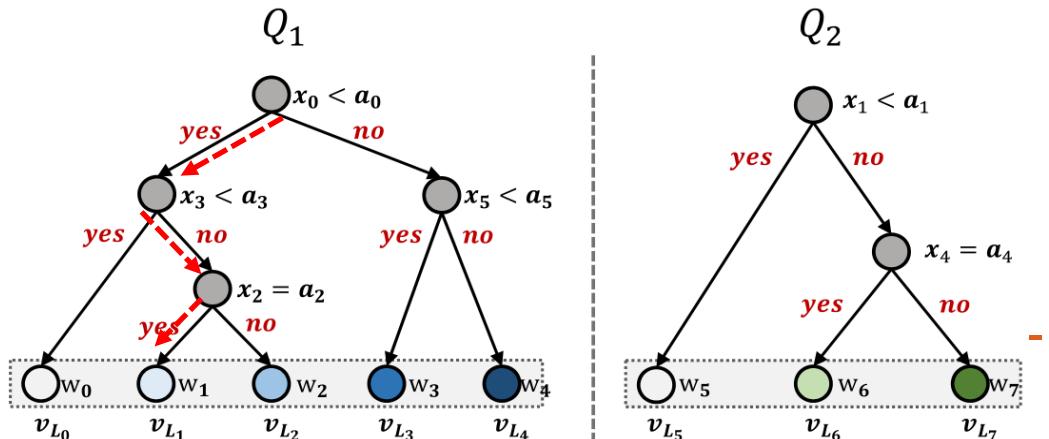
- Simple & Easy to interpret
- Easy to do online learning

Cons:

- Features are independent
- Need manual feature engineering to design cross features.

### Decision Tree (DT):

- A node splits a feature into decision edges based on its value.
- A **path** from root to leaf forms a decision rule (i.e., **cross feature**).
- Leaf node stores **prediction value**.



### Gradient Boosting Decision Trees (GBDT):

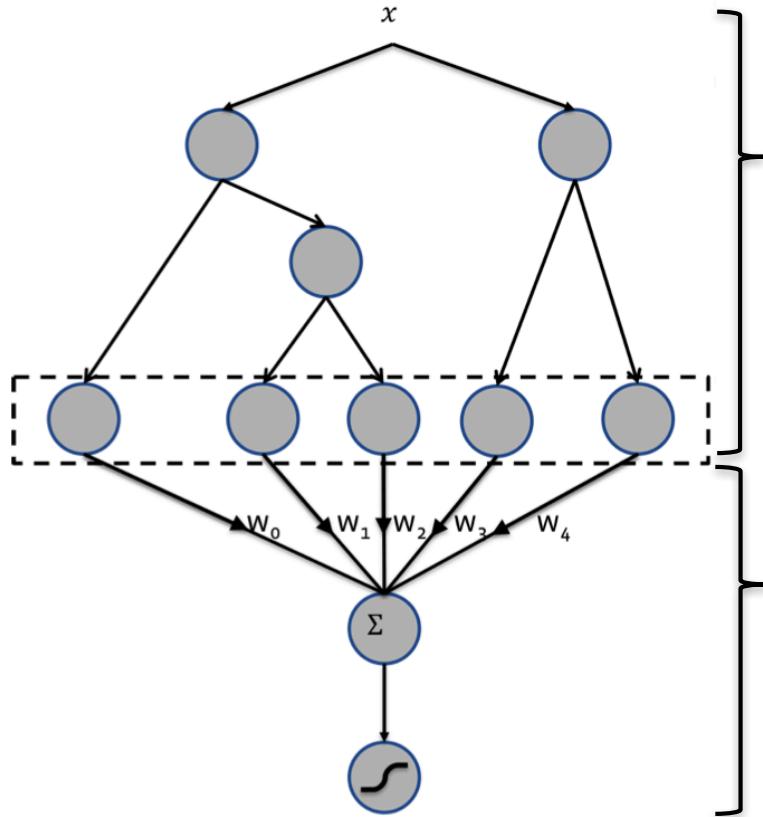
- Build multiple trees
- Combine predictions of multiple trees in an **additive** way

$$\hat{y}_{GBDT}(\mathbf{x}) = \sum_{s=1}^S \hat{y}_{DT_s}(\mathbf{x}),$$

Prediction of the  $s$ -th tree

Can capture more complex decision patterns than a single tree.

- Early Facebook CTR prediction solution (He et al, ADKDD'14)



GBDT: converts feature vector to multiple cross features  
 - Acts as a **non-linear feature transformer**.

LR: **reassigns the weights** of cross features for a prediction

### Pros:

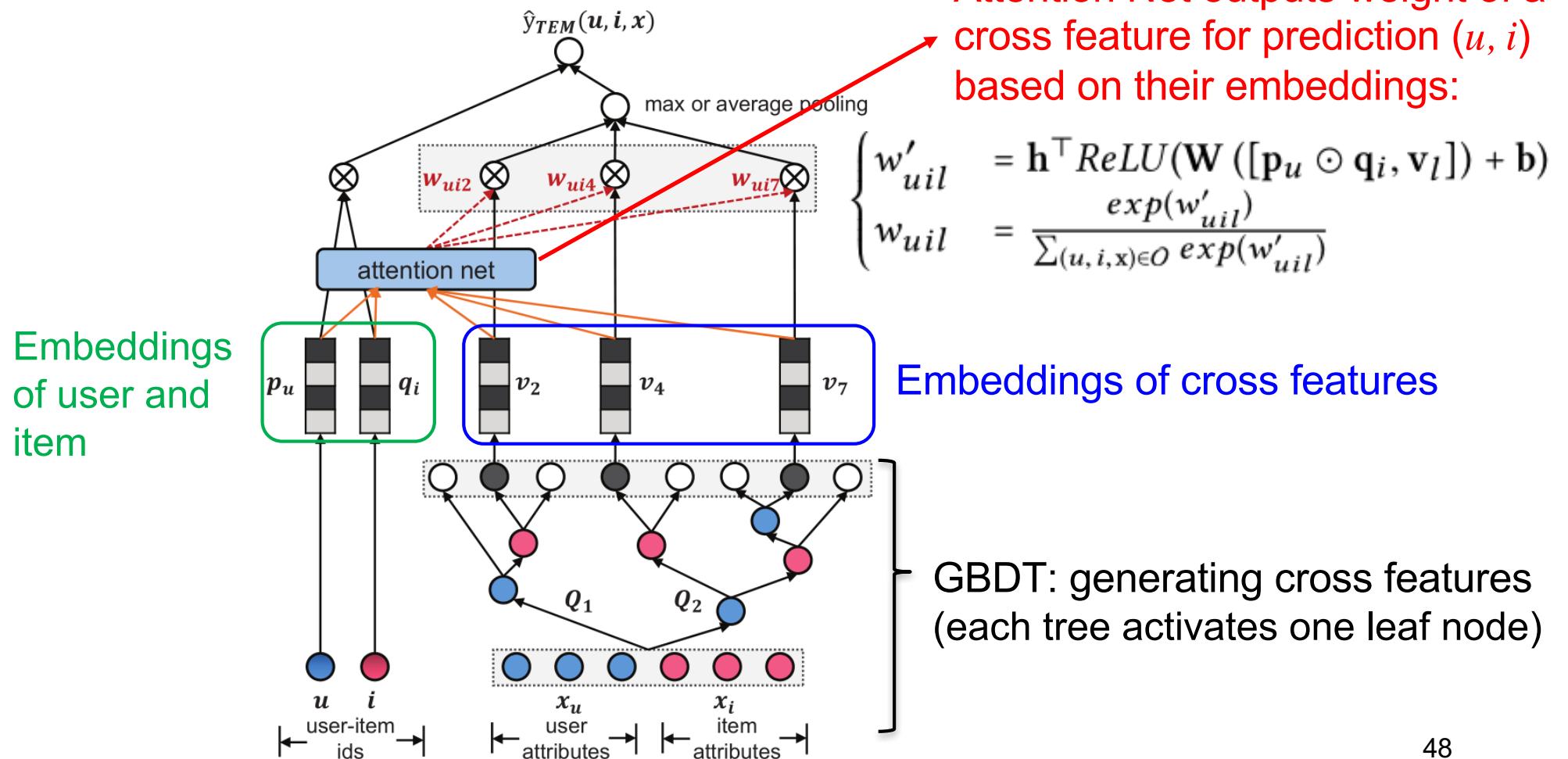
- Alleviate the need for manual feature engineering

### Cons:

- Limited expressiveness: the weight of a cross feature is unchanged for all predictions.

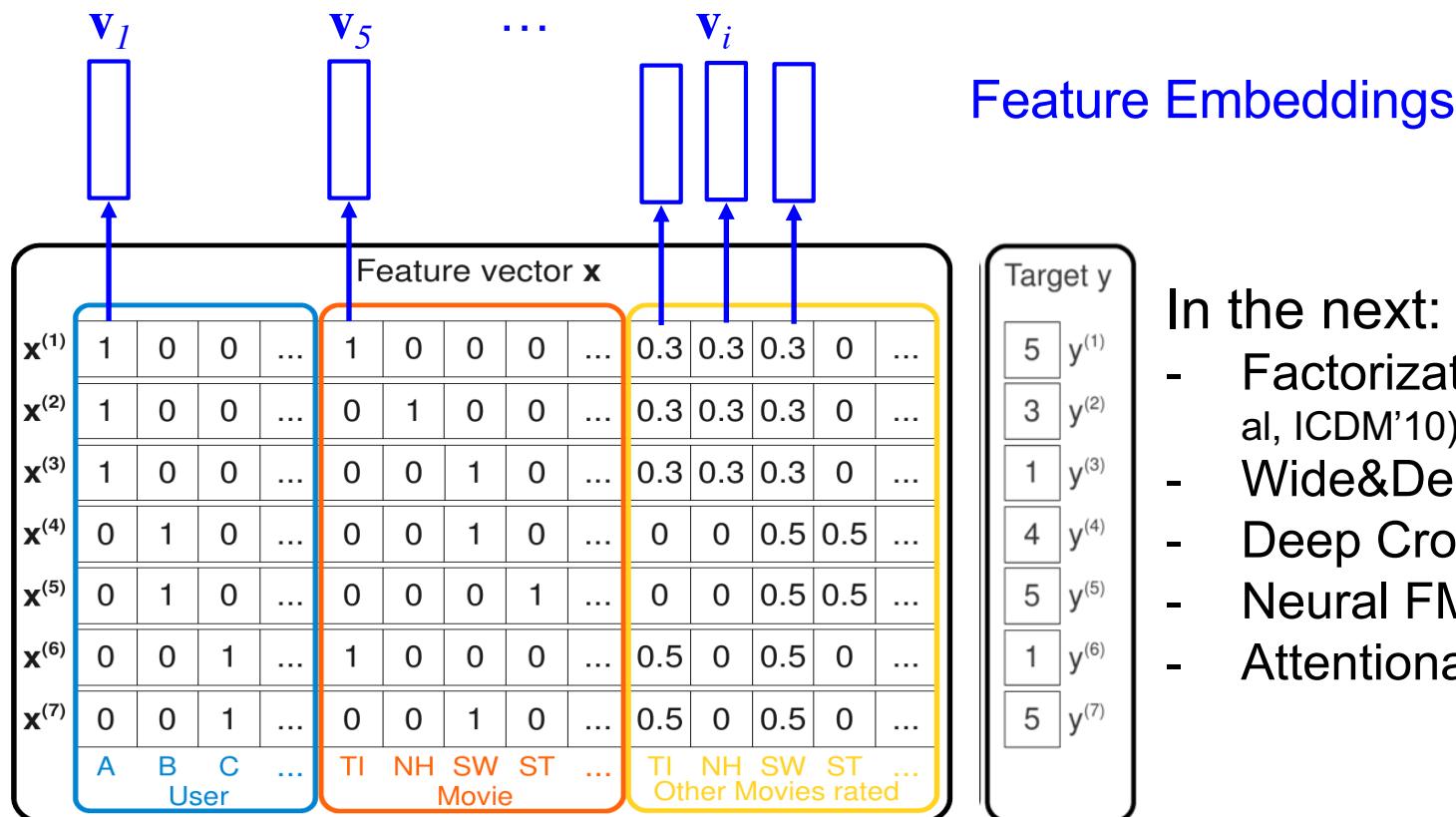
## GBDT + attention net

- Tree-enhanced Embedding Model (TEM, Wang et al, WWW'17) **adapts the weights** of cross features by using neural attention network.



## Embedding-based models

- Learning the semantics of features in vector space.
  - Each feature is associated with an embedding vector.
  - Model prediction is a function of embeddings, rather than raw features.



In the next:

- Factorization Machine (Rendle et al, ICDM'10)
- Wide&Deep (Cheng et al, Recsys'16)
- Deep Crossing (Shan et al, KDD'16)
- Neural FM (He et al, SIGIR'17)
- Attentional FM (Xiao et al, IJCAI'17)

## Factorization machine

- Extend LR by modeling **pairwise interactions** between feature embeddings with **inner product**: Only nonzero features are considered

$$\hat{y}(\mathbf{x}) = w_0 + \underbrace{\sum_{i=1}^p w_i x_i}_{\text{First-order: Linear Regression}} + \underbrace{\sum_{i=1}^p \sum_{j>i}^p \langle \mathbf{v}_i, \mathbf{v}_j \rangle}_{\text{Second-order: pair-wise interactions between features}} \boxed{x_i x_j}$$

First-order: Linear Regression

Second-order: pair-wise interactions between features

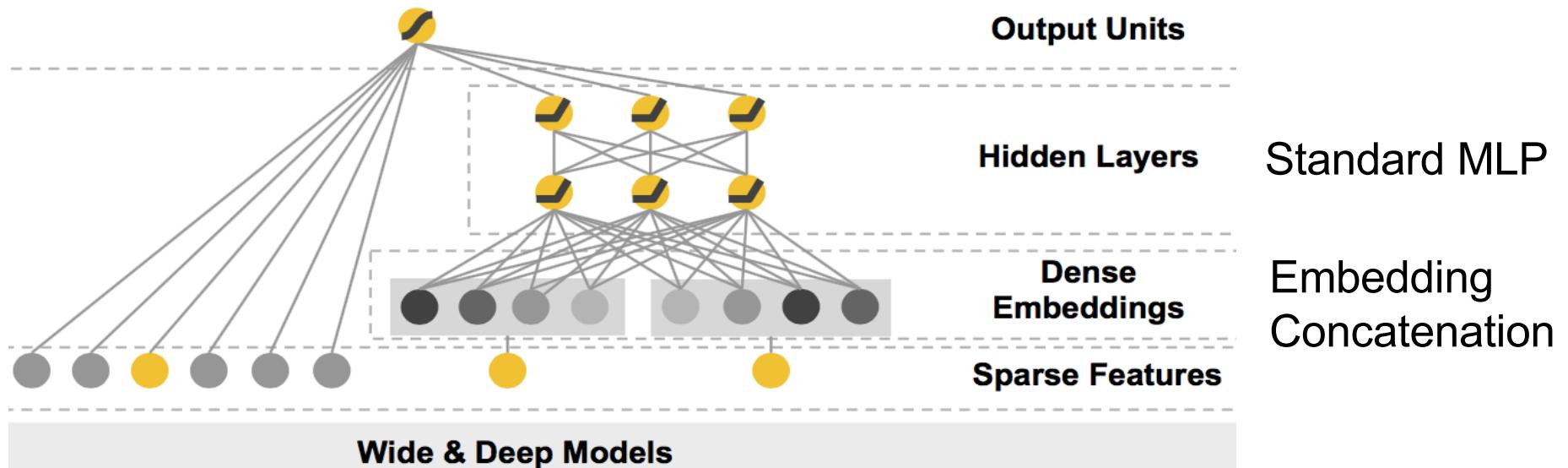
- Note: self-interaction is not included:  $\cancel{\langle \mathbf{v}_i, \mathbf{v}_i \rangle}$ .
  - Example: 

Publisher (P)	Advertiser (A)	Gender (G)
ESPN	Nike	Male

$$y = w_{\text{ESPN}} + w_{\text{Nike}} + w_{\text{Gender}} + \langle \mathbf{v}_{\text{ESPN}}, \mathbf{v}_{\text{Nike}} \rangle + \langle \mathbf{v}_{\text{ESPN}}, \mathbf{v}_{\text{Male}} \rangle + \langle \mathbf{v}_{\text{Nike}}, \mathbf{v}_{\text{Male}} \rangle$$
- |   |  |
|---|--|
| <b>Pros:</b>  | <b>Cons:</b>   |
| <ul style="list-style-type: none"> <li>- Feature interactions are learned automatically.</li> </ul> | <ul style="list-style-type: none"> <li>- Only 2nd-order feature interactions. (inefficient for higher order interactions)</li> </ul> |

## Wide & deep

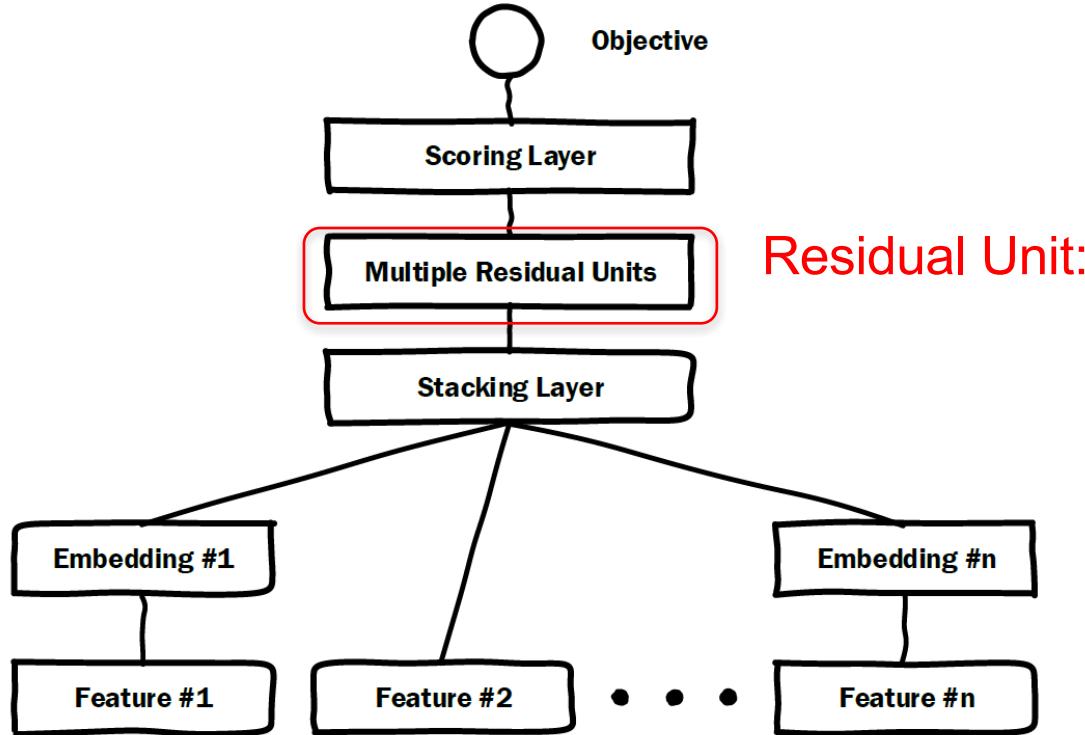
### Google's App Recommender Ranking Solution in 2016



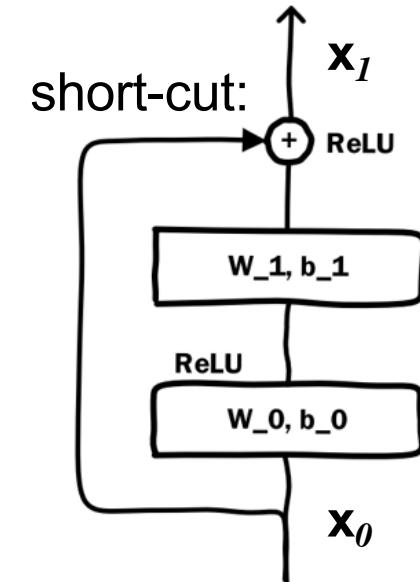
- The wide part is linear regression for memorizing seen feature interactions, which requires careful engineering on cross features.  
e.g.,  $\text{AND}(\text{gender=female}, \text{language=en})$  is 1 iff both single features are 1
- The deep part is DNN for generalizing to unseen feature interactions.  
Feature interactions are captured in an implicit way.

## Deep crossing

Microsoft's CTR Prediction Solution in 2016:



Residual Unit:



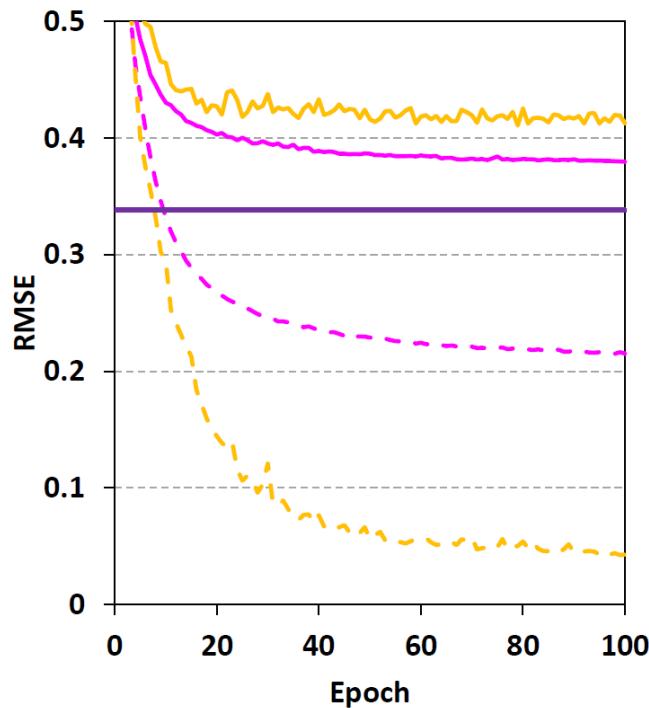
The main difference from Wide&Deep is the use of residual layers, which allow deeper network to be built (~10 layers).

## Empirical evidence

- However, when only **raw features** are used, Wide&Deep and DeepCross don't perform well in learning feature interactions.

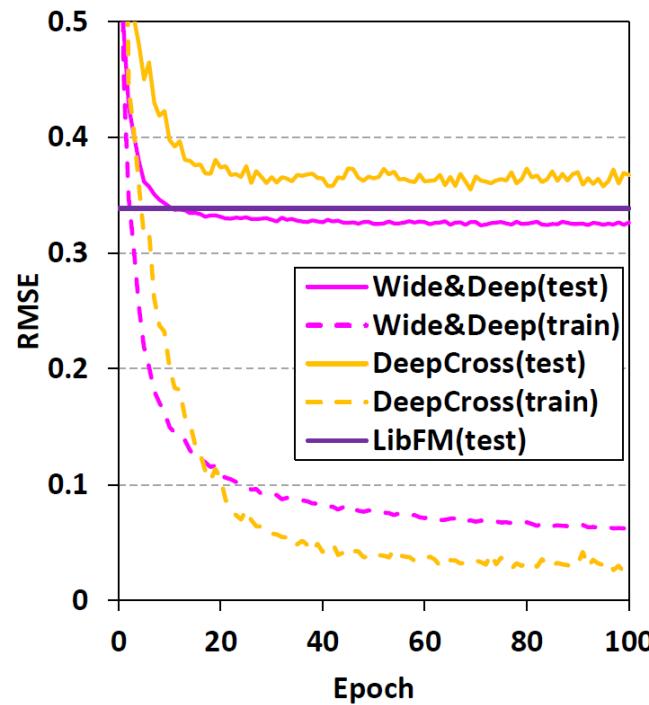
Solid line: testing loss;

Dashed line: training loss



(a) Random initialization

With random initialization, two deep methods underperform FM.



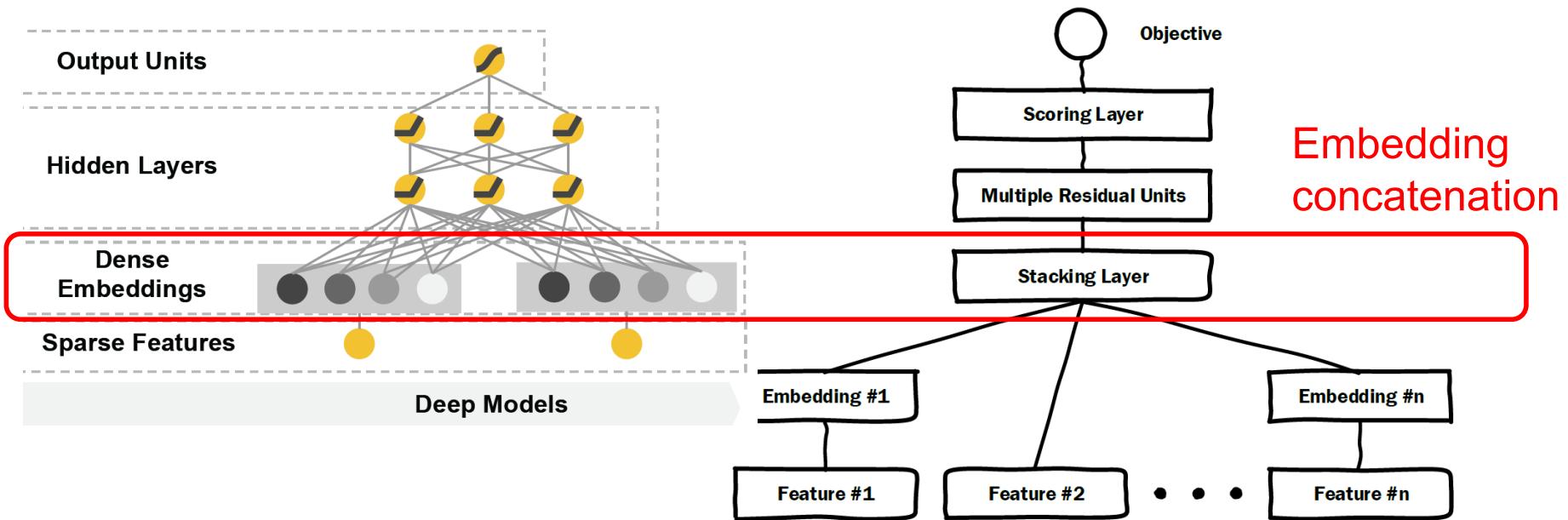
(b) FM as pre-training

With FM embeddings as pre-training, Wide&Deep slightly outperforms FM.

Some issues of DL methods:  
*Easy to overfit*  
*Hard to train well*  
*Need good init.*

## Why is DNN ineffective?

Besides optimization difficulties, one reason is in model design:

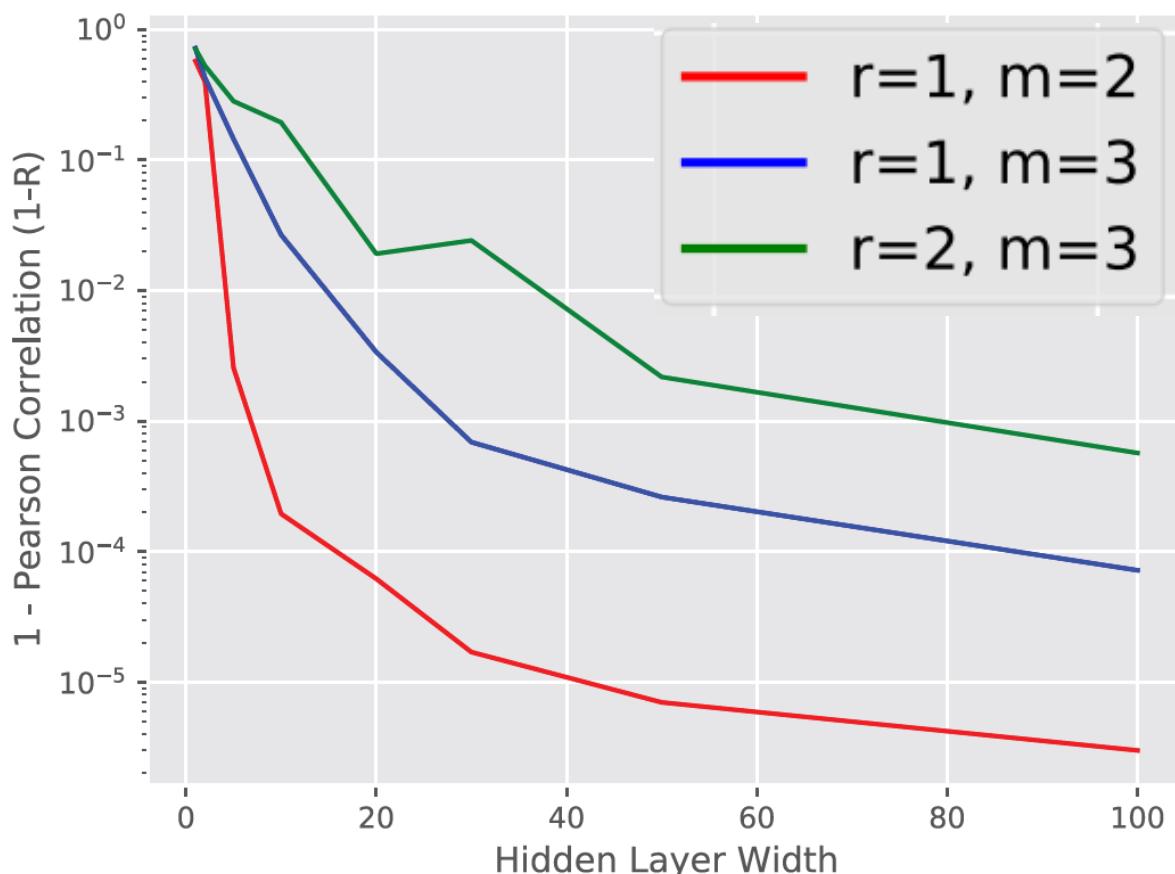


1. Embedding concatenation carries **little information** about feature interactions in the low level!
2. The structure of Concat+MLP is ineffective in learning the **multiplicative relation** (Beutel et al, WSDM'18).

## DNN is weak in capturing multiplicative relation

- Evidence from Google researchers (Beutel et al, WSDM'18)
  - Setting: generate low-rank data, and use one-layer MLP to fit it

r: rank size; m: data dimension (2 -> matrix; 3 -> 3D tensor).



MLP can learn low-rank relation, but is **inefficient** in doing so!

- Need to use 100 neurons to fit a rank-1 matrix.

Insight: need to augment DNN with multiplicative relation modeling

## Neural factorization machine

- Neural FM “deepens” FM by placing DNN above **second-order interaction modeling**.
  - Interaction is modeled with **multiplication** between embeddings

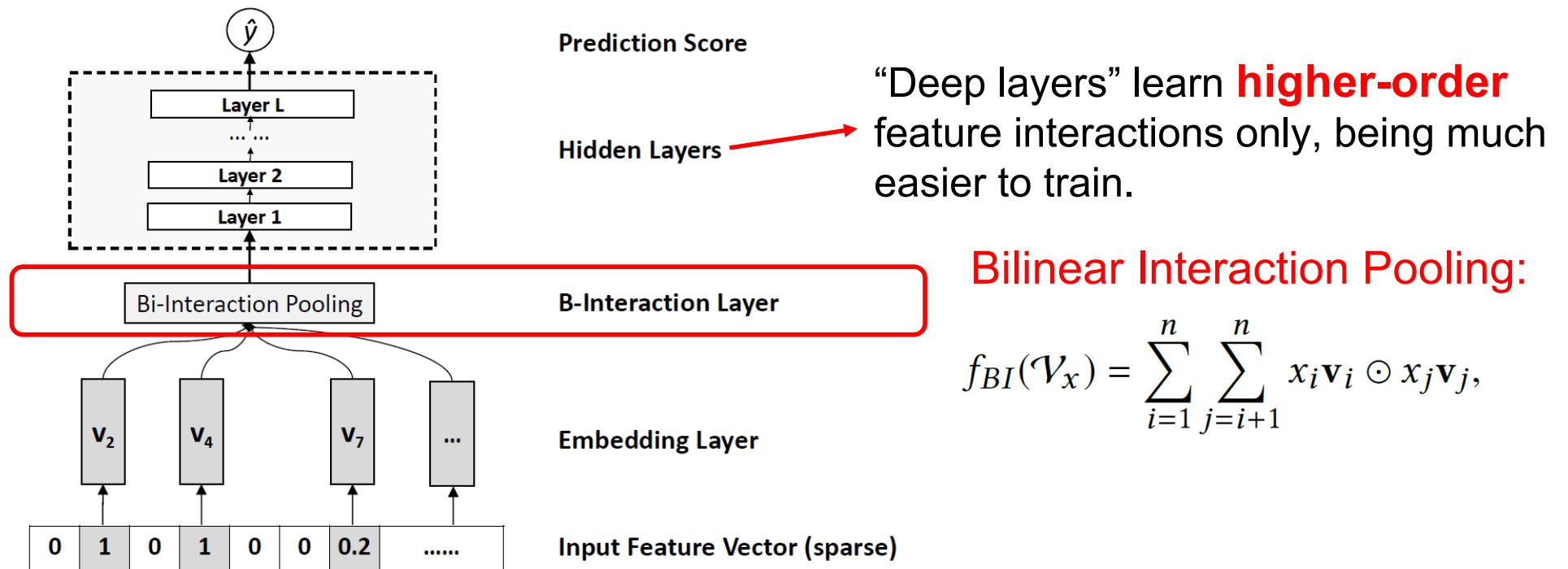


Figure 2: Neural Factorization Machines model (the first-order linear regression part is not shown for clarity).

## Empirical evidence

All methods are fed into raw features without any feature engineering

### Task #1: Context-aware App Usage Prediction

- Frappe data: instance #: 288,609, feature #: 5,382

### Task #2: Personalized Tag Recom

- MovieLens data: Inst #: 2,006,859, Feat #: 90,445

**Table: Parameter # and testing RMSE at embedding size 128**

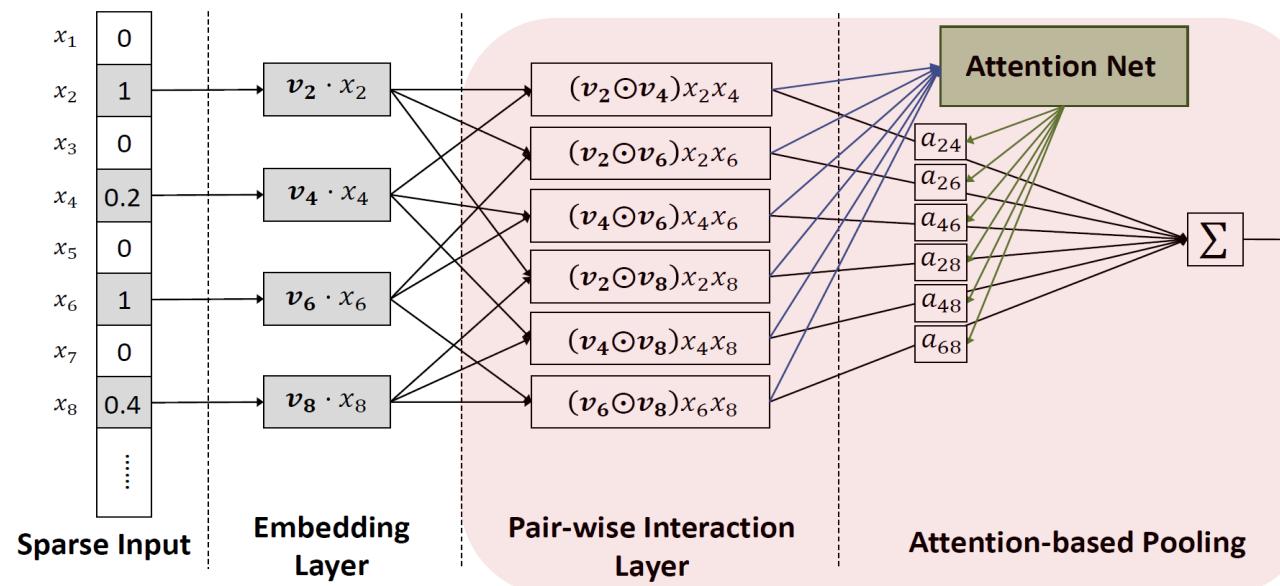
Method	Frappe		MovieLens	
	Param#	RMSE	Param#	RMSE
Logistic Regression	5.38K	0.5835	0.09M	0.5991
FM	1.38M	0.3385	23.24M	0.4735
High-order FM	2.76M	0.3331	46.40M	0.4636
Wide&Deep (3 layers)	4.66M	0.3246	24.69M	0.4512
DeepCross (10 layers)	8.93M	0.3548	25.42M	0.5130
<b>Neural FM (1 layer)</b>	<b>1.45M</b>	<b>0.3095</b>	<b>23.31M</b>	<b>0.4443</b>

1. Embedding methods learn interactions, better than simple linear models
2. Deep embedding methods:  
**Wide&Deep = Concat+3 layers**  
**DeepCross = Concat+10 layers**
3. Neural FM  
**= BI pooling + 1 layer**  
Shallower but outperforming existing deeper methods with less parameters.

Code: [github.com/hexiangnan/neural\\_factorization\\_machine](https://github.com/hexiangnan/neural_factorization_machine)

## Attentional FM

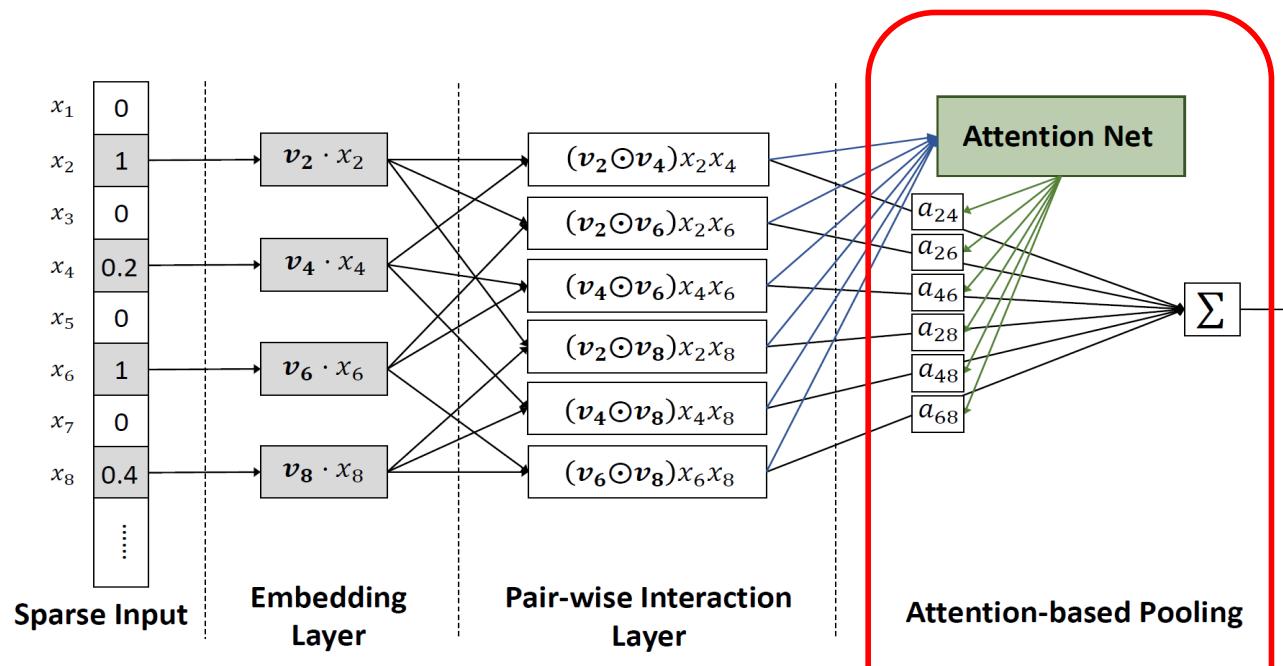
- Neural FM treats all second-order feature interactions as **contributing equally**.
  - However, some interactions may not be important.
- Attentional FM uses an **attention network** to learn the **weight** of a feature interaction.



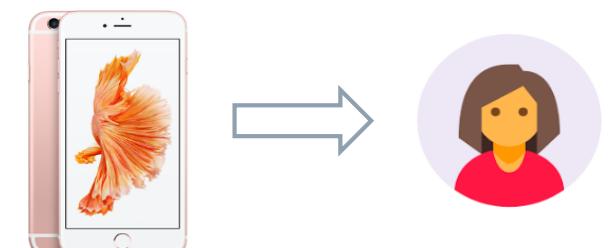
$$f_{ABI}(\mathcal{V}_x) = \sum_{i=1}^n \sum_{j=i+1}^n (\mathbf{x}_i \mathbf{v}_i \odot \mathbf{x}_j \mathbf{v}_j) a_{ij}$$

$$\begin{aligned} a'_{ij} &= \mathbf{h}^T \text{ReLU}(\mathbf{W}(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j + \mathbf{b}), \\ a_{ij} &= \frac{\exp(a'_{ij})}{\sum_{(i,j) \in \mathcal{R}_x} \exp(a'_{ij})}, \end{aligned}$$

The attention scores can be used to select the most predictive second-order feature interactions as explanations.



Example: **explainable recommendation** with second-order cross features:



<Female, Age 20>  
 <Age 20, iPhone>  
 <Female, Color Pink>

.....

## Empirical evidence

All methods are fed into raw features without any feature engineering

### Task #1: Context-aware App Usage Prediction

- Frappe data: instance #: 288,609, feature #: 5,382

### Task #2: Personalized Tag Recom

- MovieLens data: Inst #: 2,006,859, Feat #: 90,445

**Table: Parameter # and testing RMSE at embedding size 128**

Method	Frappe		MovieLens	
	Param#	RMSE	Param#	RMSE
Logistic Regression	5.38K	0.5835	0.09M	0.5991
FM	1.38M	0.3385	23.24M	0.4735
High-order FM	2.76M	0.3331	46.40M	0.4636
Wide&Deep (3 layers)	4.66M	0.3246	24.69M	0.4512
DeepCross (10 layers)	8.93M	0.3548	25.42M	0.5130
<b>Neural FM (1 layer)</b>	<b>1.45M</b>	<b>0.3095</b>	<b>23.31M</b>	<b>0.4443</b>
<b>AFM (0 layer)</b>	<b>1.45M</b>	<b>0.3102</b>	<b>23.26M</b>	<b>0.4325</b>

Codes: [github.com/hexiangnan/neural\\_factorization\\_machine](https://github.com/hexiangnan/neural_factorization_machine)

AFM without hidden layers can even be better than NFM with 1 hidden layer.

Adding hidden layers to AFM further improves.

## Summary of recommendation

- Candidate Generation and Ranking are two major components for E-Commerce recommender systems
- For Candidate Generation:
  - Efficiency and recall are the main concern
  - The key to kNN-based solution is how to learn good features for user and item
- For Ranking:
  - Precision is the main concern.
  - The key to feature-based models is how to effectively learn feature interactions.
- Currently, the two steps are separated tuning.
  - A future direction is to jointly optimize both steps
  - E.g., using rewards from ranking stage to improve candidate generation.

## References

- Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In Recsys 2016.
- Dhruv Arya, Ganesh Venkataraman, Aman Grover, and Krishnaram Kenthapadi. Candidate Selection for Large Scale Personalized Search and Recommender Systems. In SIGIR 2017.
- James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The YouTube Video Recommendation System. In Recsys 2010.
- Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2018. Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time. In WWW 2018.
- Zihan Wang, Ziheng Jiang, Zhaochun Ren, Jiliang Tang, and Dawei Yin. A Path-constrained Framework for Discriminating Substitutable and Complementary Products in E-commerce. In WSDM 2018.
- Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In SIGIR 2016.
- Santosh Kabbur, Xia Ning, and George Karypis. Fism: factored item similarity models for top-n recommender systems. In KDD 2013.
- Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In KDD 2008.
- Yehuda Koren. Collaborative filtering with temporal dynamics. In KDD 2009.
- Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized Markov chains for next-basket recommendation. In WWW 2010.
- Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized Markov chains for next-basket recommendation. In VLDB 2012.
- Aditya Krishna Menon, Krishna-Prasad Chitrapura, Sachin Garg, Deepak Agarwal, and Nagaraj Kota. Response prediction using collaborative filtering with hierarchies and side-information. In KDD 2011.
- Xue Geng, Hanwang Zhang, Jingwen Bian, and Tat-Seng Chua. Learning image and user features for recommendation in social networks. In ICCV 2015.

## References

- Ruining He and Julian McAuley. VBPR: visual Bayesian Personalized Ranking from implicit feedback. In AAAI 2016.
- Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In SIGIR 2017.
- Xu Chen et al. Visually Explainable Recommendation. Arxiv 2018.
- Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In UAI 2009.
- Weinan Zhang, Tianqi Chen, Jun Wang, and Yong Yu. Optimizing top-n collaborative filtering via dynamic negative item sampling. In SIGIR 2013.
- Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, Jure Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In KDD 2018.
- Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In WWW 2017.
- Steffen Rendle. Factorization machines. In ICDM 2010.
- Xiang Wang, Xiangnan He, Fuli Feng, Liqiang Nie, and Tat-Seng Chua. TEM: Tree-enhanced Embedding Model for Explainable Recommendation. In WWW 2018.
- Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. Practical Lessons from Predicting Clicks on Ads at Facebook. In ADKDD 2014.
- Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson et al. Wide & deep learning for recommender systems. In DLRS 2016.
- Ying Shan, T. Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and J. C. Mao. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In KDD 2016.
- Xiangnan He, and Tat-Seng Chua. Neural factorization machines for sparse predictive analytics. In SIGIR 2017.
- Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. Attentional factorization machines: Learning the weight of feature interactions via attention networks. IJCAI 2017.