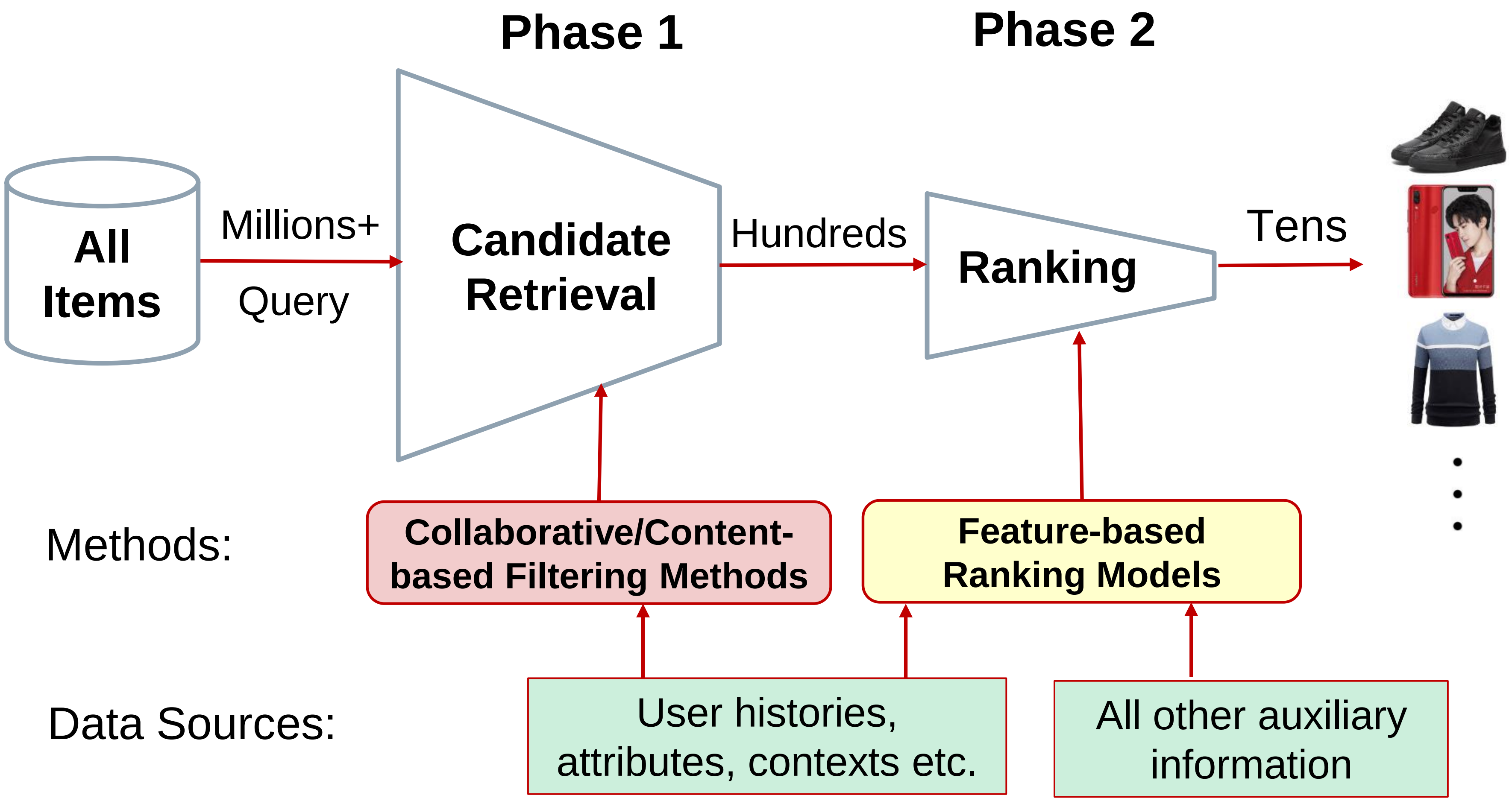
第十四次课：推荐系统简介

何向南
大数据学院
hexn@ustc.edu.cn

2020.6.12

Tutorial given in SIGIR 2018, Ann Arbor, Michigan, July 8-12, 2018

Recommender System Overview



(Covington et al, Recsys'16)

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)

(Arya et al, SIGIR'17)

Candidate Generation Overview

- 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

- 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:

Co-occurrence count

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

Normalization function, e.g.,

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

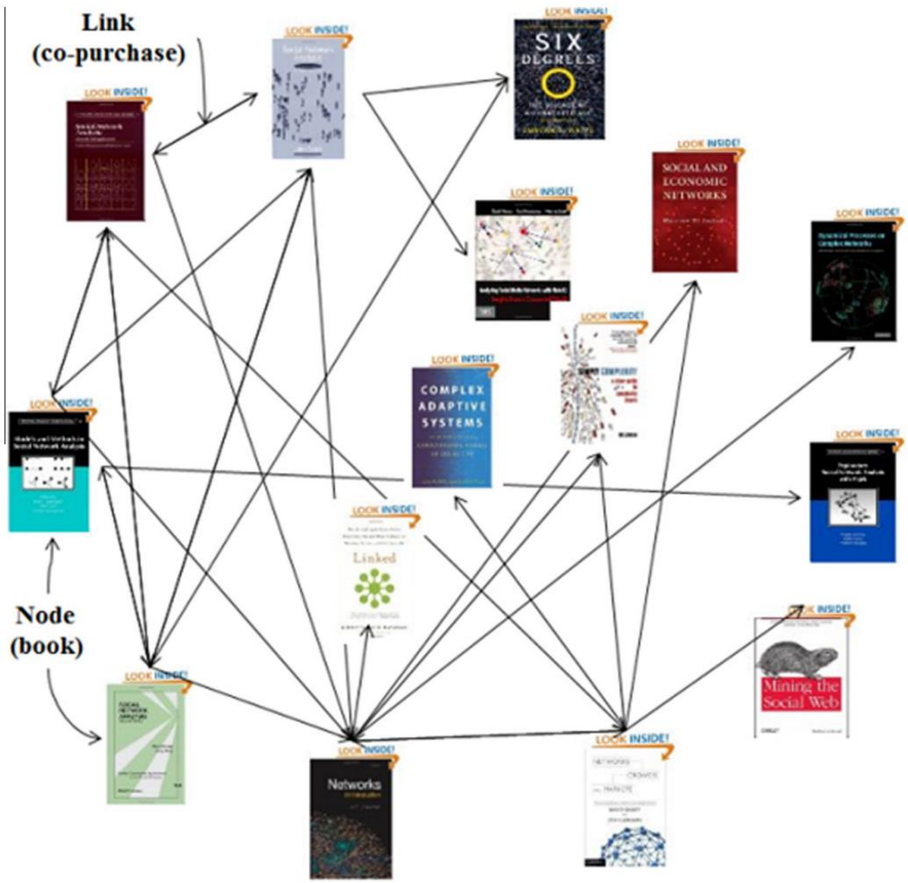
Product of items' global popularity

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

(Davidson et al, Recsys'10)

Find Similar Items (item graph)

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



(Leem et al, ESA'14)

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.

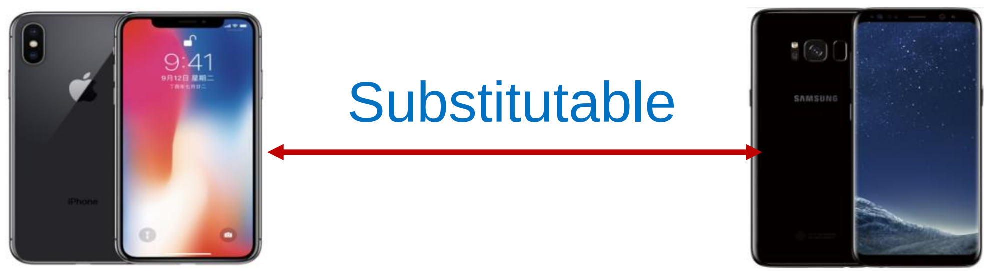
Find Similar Items (graph random walk)

- 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

(Jure et al, WWW'18)

Beyond Similar Items – Compl. & Subst.

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



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

(Wang et al, WSDM'18)

Benefits of Substitutes and Complements

- 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 stickness

(Wang et al, WSDM’18)

Learning Substitutes and Complements

- 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 (Wang et al, WSDM'18)

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

Context vector of item *j*

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

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

Target vector of item *i*

  - Parameters are learned by maximum likelihood:

$$L(Y|V,V') = -log(P(Y|V,V'))$$
$$= \boxed{-\sum_{(i,j)\in\mathcal{E}_P} log(\sigma(v_i^T \cdot v_j'))}$$
$$\boxed{-N \cdot E_{j'\sim P_n(w)}[log(\sigma(-v_i^T \cdot v_{j'}'))]},$$

Positive examples

Sampled negative examples

Basic Model + Relation Type (Wang et al, WSDM'18)

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

Relation type vector

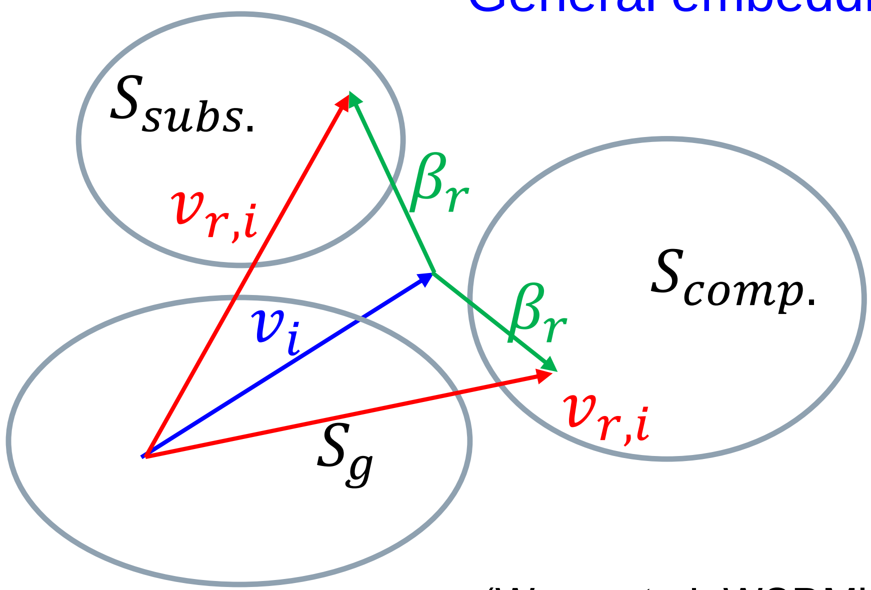
$$P(z_{i,j,r}) = \sigma(\boxed{v_{r,i}^T \cdot v'_{r,j}}),$$

$$v_{r,i} = \boxed{v_i} + \boxed{\beta_r} \odot \boxed{v_i}$$

$$v'_{r,i} = \boxed{v'_i} + \boxed{\beta_r} \odot \boxed{v'_i},$$

Item vector in relation space

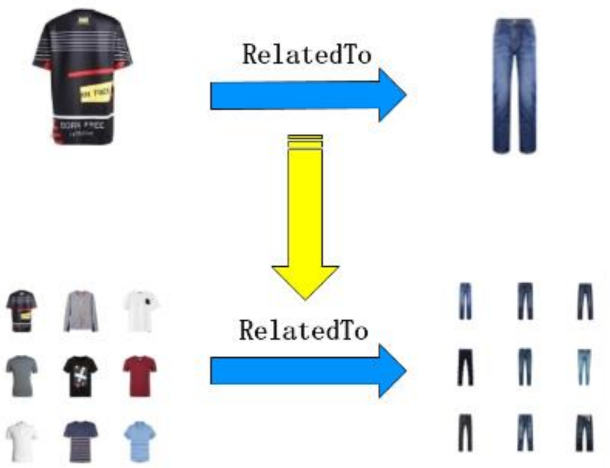
General embedding



(Wang et al, WSDM'18)

Basic Model + Relation Type + Constraints
(Wang et al, WSDM'18)

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



$(i, subst, j) => (i, subst, \boxed{Cat(j)})$

Items in the same category as *j*

T-norm fuzzy logic:

$$I(f_a \Longrightarrow f_b) = I(f_a) \cdot I(f_b) - I(f_a) + 1$$

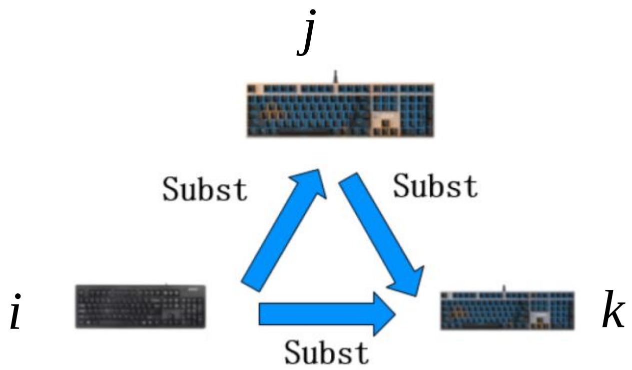
If (i, r, j) && k is in $Cat(j)$

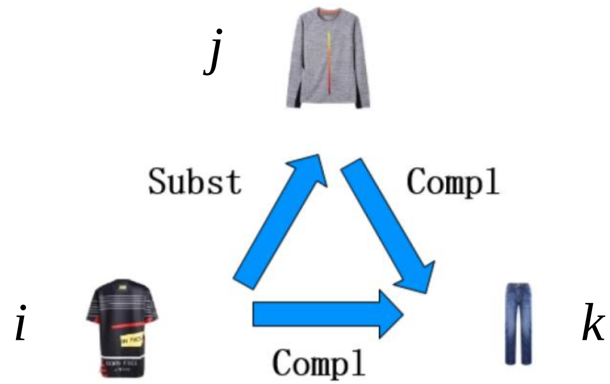
Then the probability that (i, r, k) is a **positive example** is:

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

Basic Model + Relation Type + Constraints
(Wang et al, WSDM'18)

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





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

Similarly, there are two more such constraints:

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

Basic Model + Relation Type + Constraints
(Wang et al, WSDM'18)

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

$$\begin{cases} (i, subst, j) \wedge (j, subst, k) => (i, subst, k) \\ (i, subst, j) \wedge (j, compl, k) => (i, compl, k) \\ (i, compl, j) \wedge (j, compl, k) => (i, compl, k) \\ (i, compl, j) \wedge (j, subst, k) => (i, compl, k) \end{cases}$$

Generic form

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

T-norm fuzzy logic:

$$I(f_a \wedge f_b) = I(f_a) \cdot I(f_b)$$

$$I(f_a \Longrightarrow f_b) = I(f_a) \cdot I(f_b) - I(f_a) + 1$$

If (i, r_1, j) && (j, r_2, k)

Then the probability that (i, r_3, k) is a **positive example** is:

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

Basic Model + Relation Type + Constraints
(Wang et al, WSDM'18)

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

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

$$\underbrace{L(Y|V,V')}_{\text{Basic model}} + \alpha_0 \cdot \underbrace{L(Z|V,V',\boxed{\boldsymbol{\beta}})}_{\text{Basic model + relation type}} + \alpha_1 \cdot \underbrace{L(F_1|V,V',\boldsymbol{\beta})}_{\text{Basic model + relation type + category constraint}} + \alpha_2 \cdot \underbrace{L(F_2|V,V',\boldsymbol{\beta})}_{\text{Basic model + relation type + category constraint + path constraints}}$$

Short Summary on Candidate Generation

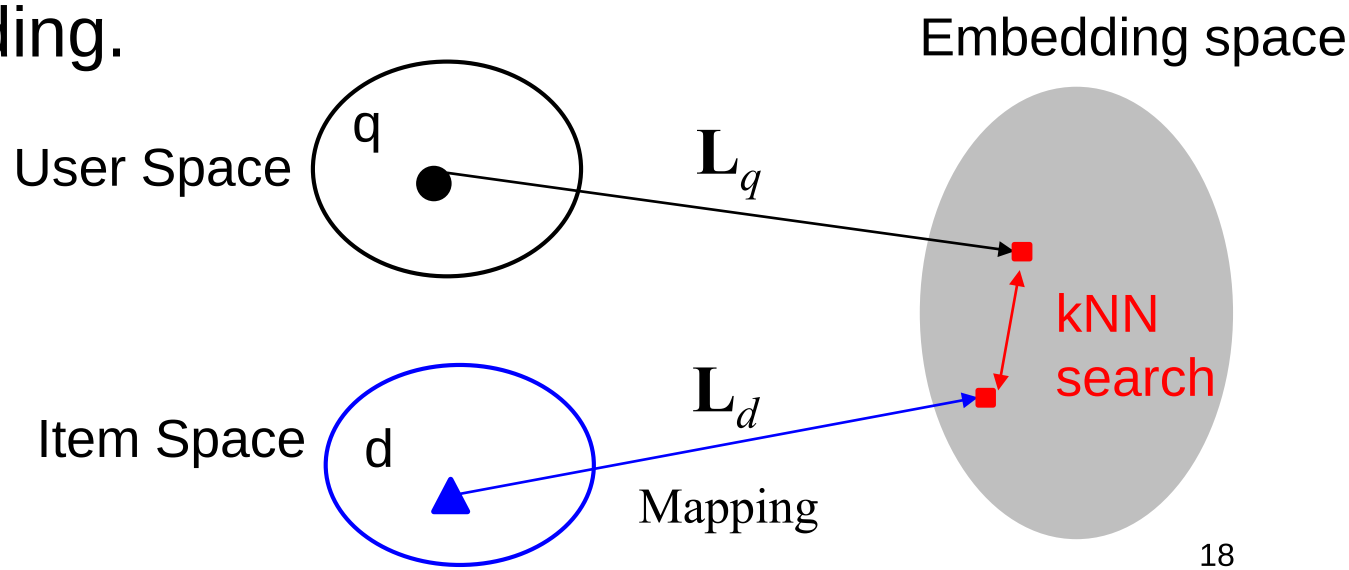
- ✓ Heuristic methods:
  - ✓ How to find similar items
  - ✓ How to find complementary and substitutable items

Short Summary on Candidate Generation

- ✓ 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

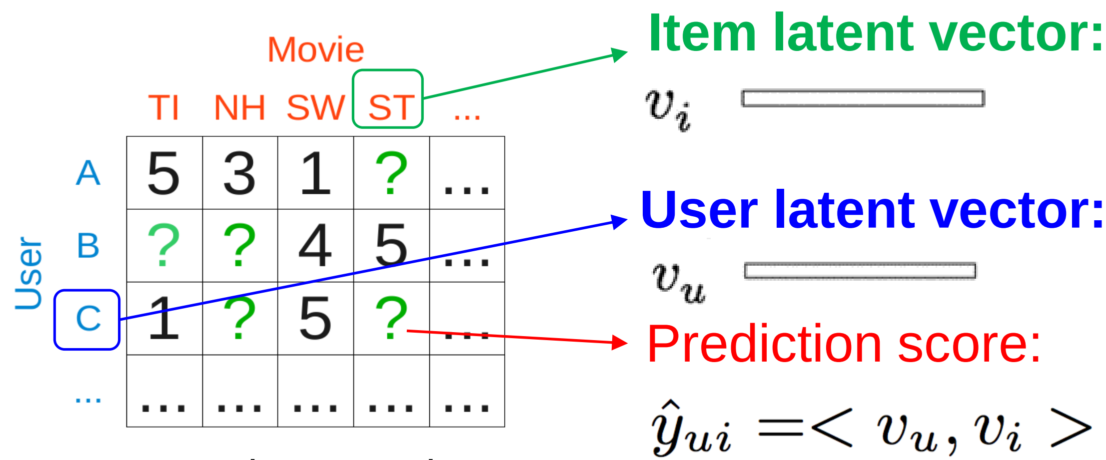
- Matrix Factorization (MF) assumes the user-item interaction matrix has a low-rank structure:

$$\mathbf{Y} \approx \boxed{\mathbf{U}}\boxed{\mathbf{V}}^T$$

Item latent matrix

User latent matrix

User	Movie	Rating
Alice	Titanic	5
Alice	Notting Hill	3
Alice	Star Wars	1
Bob	Star Wars	4
Bob	Star Trek	5
Charlie	Titanic	1
Charlie	Star Wars	5
...	...	...



User-item matrix Y

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

(He et al, SIGIR'16)

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 **U**
- Another more information-rich way for user profile is to use the rating history:

$$\hat{y}_{ui} = \Big(\underbrace{\frac{1}{|\mathcal{R}_u|^\alpha} \sum_{\boxed{j \in \mathcal{R}_u}} \mathbf{q}_j}_{\Leftrightarrow \text{ user embedding}}\Big)^T \mathbf{v}_i$$

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:

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

  - 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 decisions are context-aware:
  - Contexts: time, location, query, weather etc.

- Context-aware MF model:

$$\hat{y}_{ui} = (\overbrace{\mathbf{v}_u + \sum_{\boxed{c \in C(u)}} \underbrace{\mathbf{v}_{u,c}}_{\text{Context-aware User Embedding}}}^{\text{User Embedding}})^T \mathbf{v}_i$$

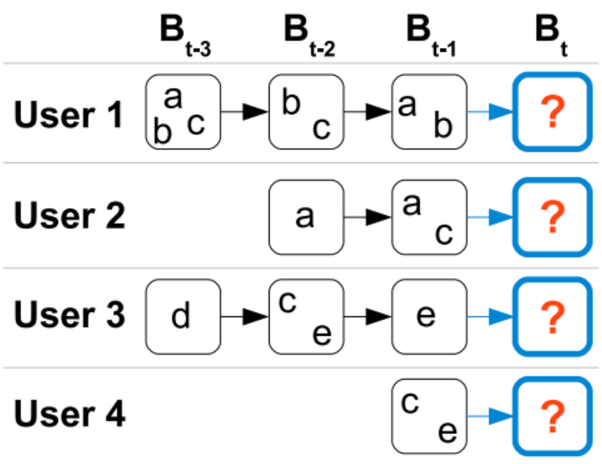
Current contexts

  - 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):

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

Estimate transition probability from last item *l* to next item *j*

Items purchased in last basket

Item Embedding + Attributes

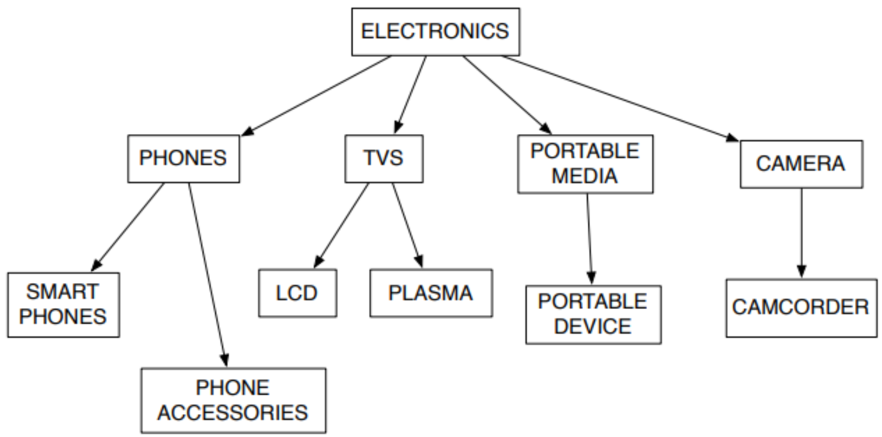
- Attributes of item side can be integrated by using similar way of embedding addition:
  - E.g., product tags, category, price

$$\hat{y}_{ui} = \mathrm{v}_u^T \big(\overbrace{\mathbf{v}_i + \sum_{a \in A(i)} \underbrace{\mathbf{v}_a}_{\text{Attribute Embedding}}}^{\text{Item Embedding}} \big)$$

- 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):

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

Embedding of intermediate node *l* in the path

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

Item Embedding + 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):

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

Parent of node i in the taxonomy

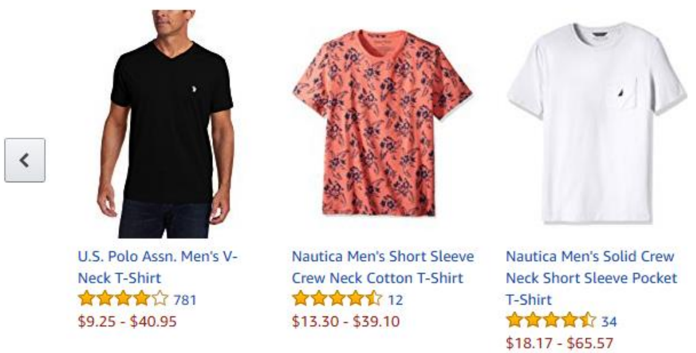
All items and taxonomy nodes

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

Item Embedding + Image

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

What user purchased:



Will user like this one?

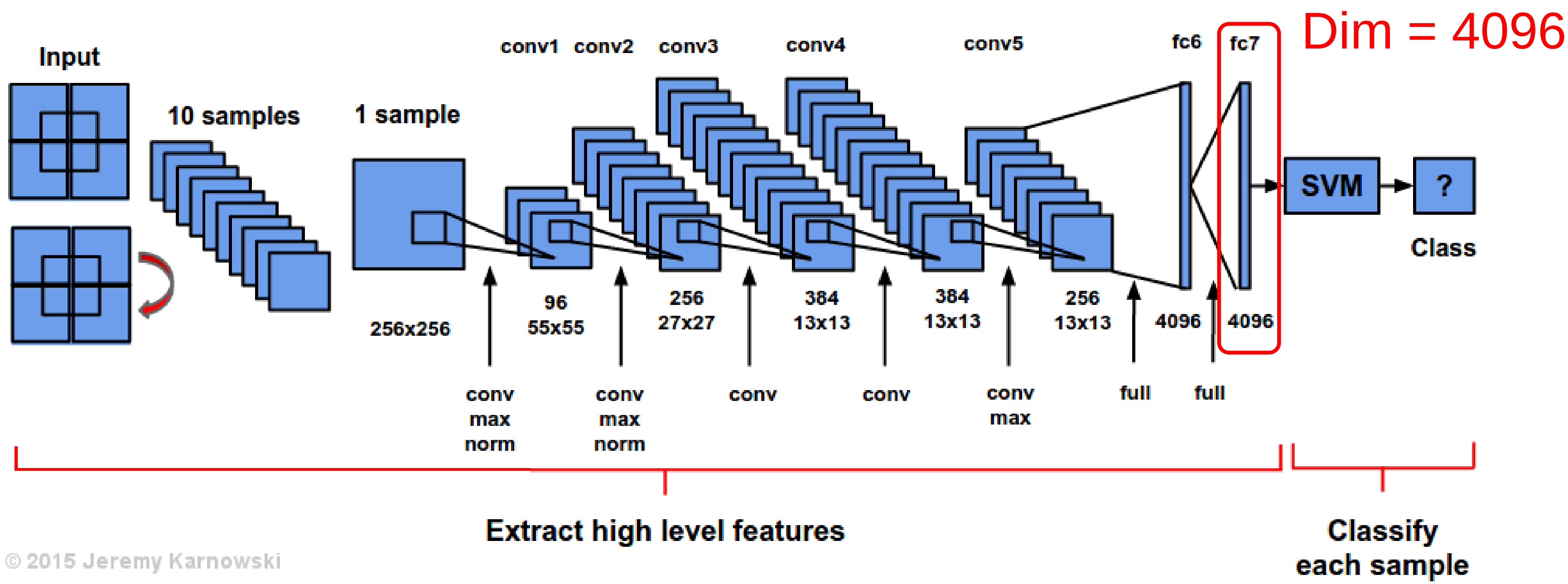
Each product = ID + image

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 Embedding + 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:

$$\hat{y}_{ui} = < \boxed{\mathbf{p}_u}, \mathbf{f}_i > = \mathbf{p}_u^T \mathbf{f}_i$$

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.

Item Embedding + Image

- 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:

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

Transformation matrix that projects CNN features to latent space

  - ➢ **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)

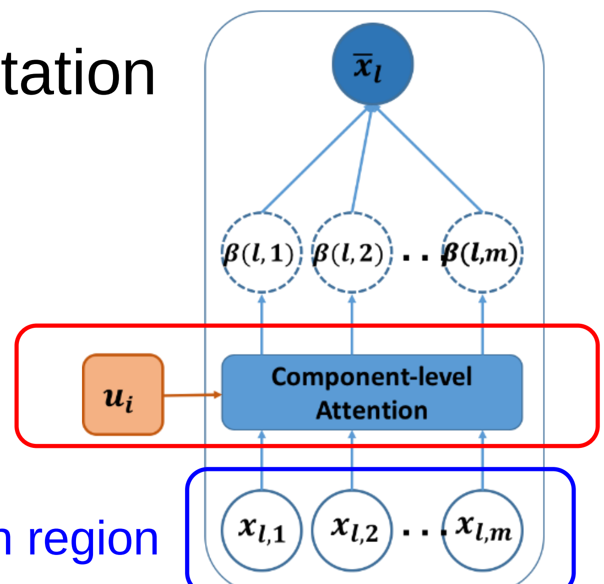
- User-sensitive Image Representation

(Chen et al, SIGIR'17)

Attention Net determines region's weight:

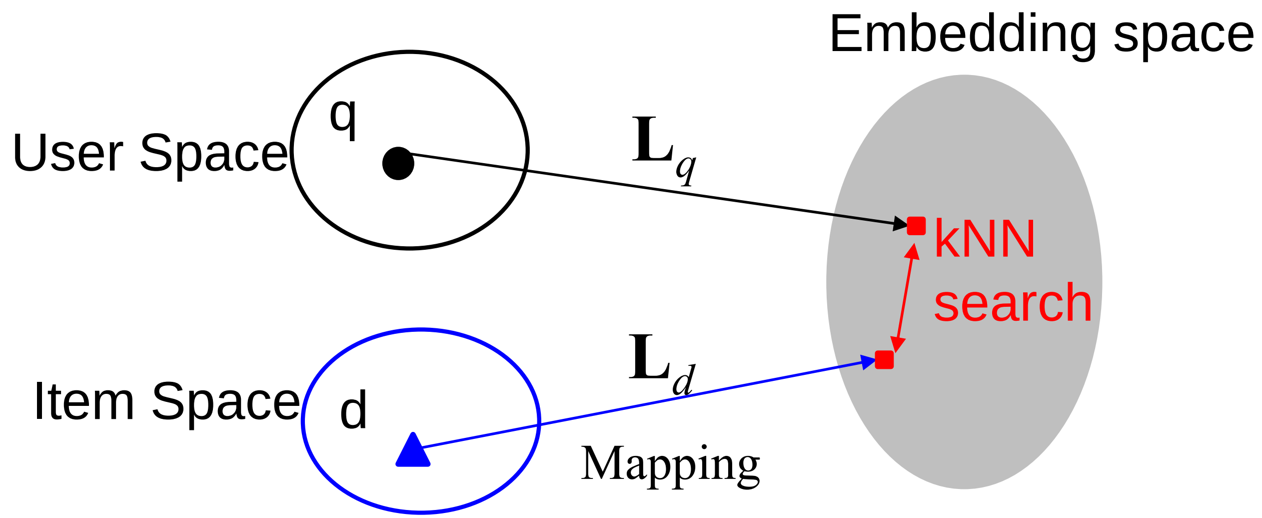
- Input: user embedding and region feature

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



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

Implicit Feedback conveys user preference implicitly:

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

Movie

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

Movie

User	TI	NH	SW	ST	...
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:

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

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/Reall/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)

sigmoid

Positive prediction

Negative prediction

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

Pairwise training examples: u prefers i over j

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

Training Procedure

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

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.

$$\mathcal{D} := \{(u,i,j) | \underbrace{i \in \mathcal{Y}_u^+}_{\text{Items purchased by } u} \wedge \underbrace{j \notin \mathcal{Y}_u^+}_{\text{Items not purchased by } u}\}$$

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 (Zhang et al, SIGIR'13)

- 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:

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

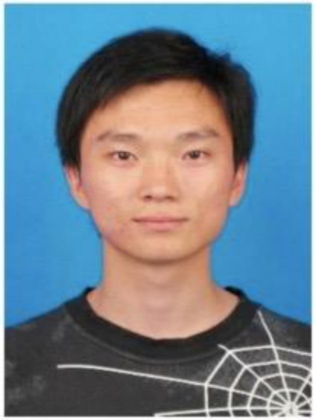
Expected minimum margin between positive prediction and negative prediction

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

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

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

业界分享

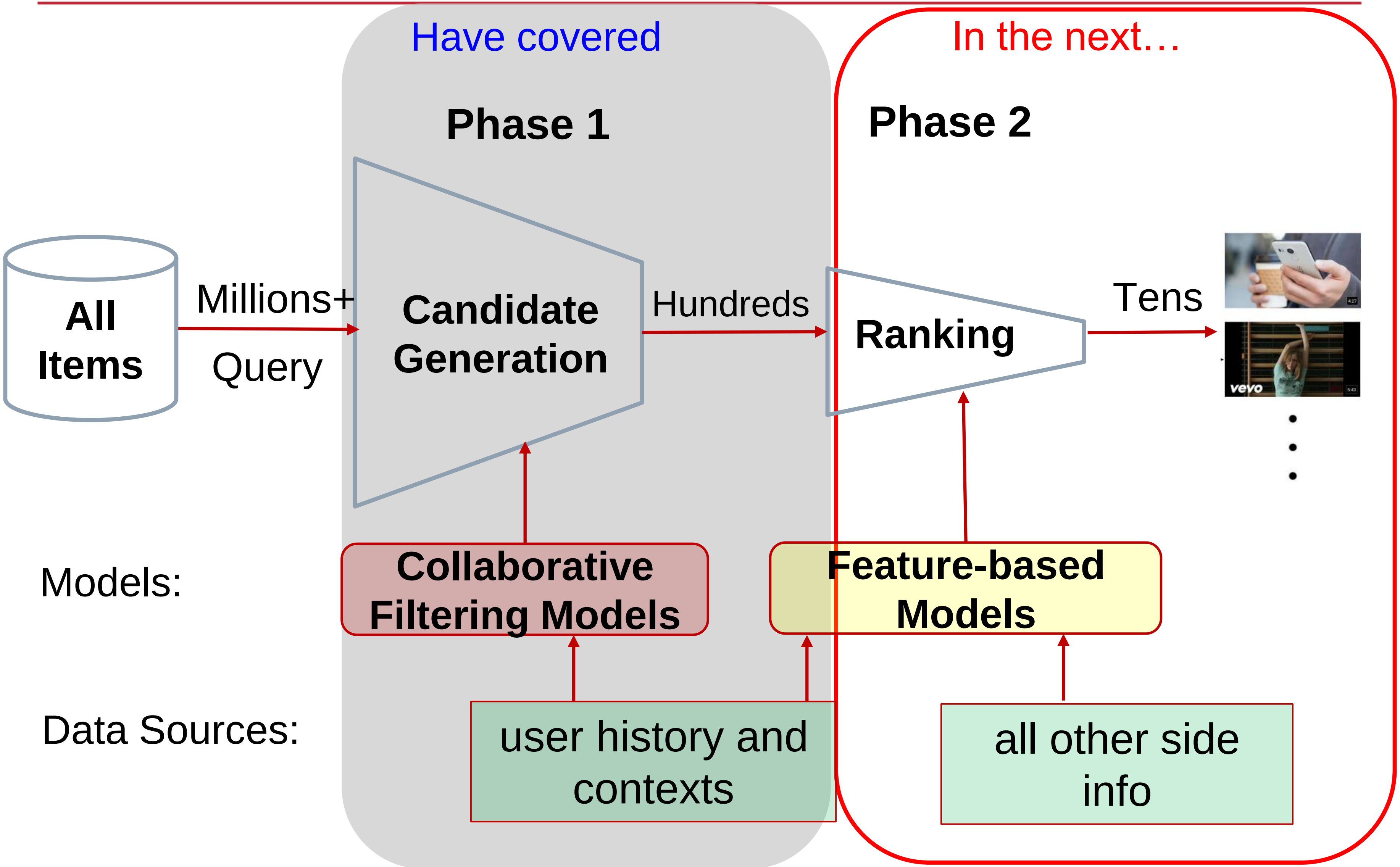


向彪（阿里花名：仁勇）
2013年中国科大计算机博士
导师：陈恩红

- 2019.7-至今 Lazada Vice President, Online Marketing & User Growth
- 2016.11-2019.7 蚂蚁金服 算法专家，人工智能部
- 2013.05-2016.11 微软 高级研发工程师，Bing搜索

- 在微软期间负责Bing的搜索相关性算法，包括深度语义匹配模型，query&doc理解，自动问答系统，知识挖掘等，在蚂蚁AI期间主要负责智能客服、安全风控、推荐营销相关算法，在Lazada负责在线营销和用户增长算法。

Recommender System Overview



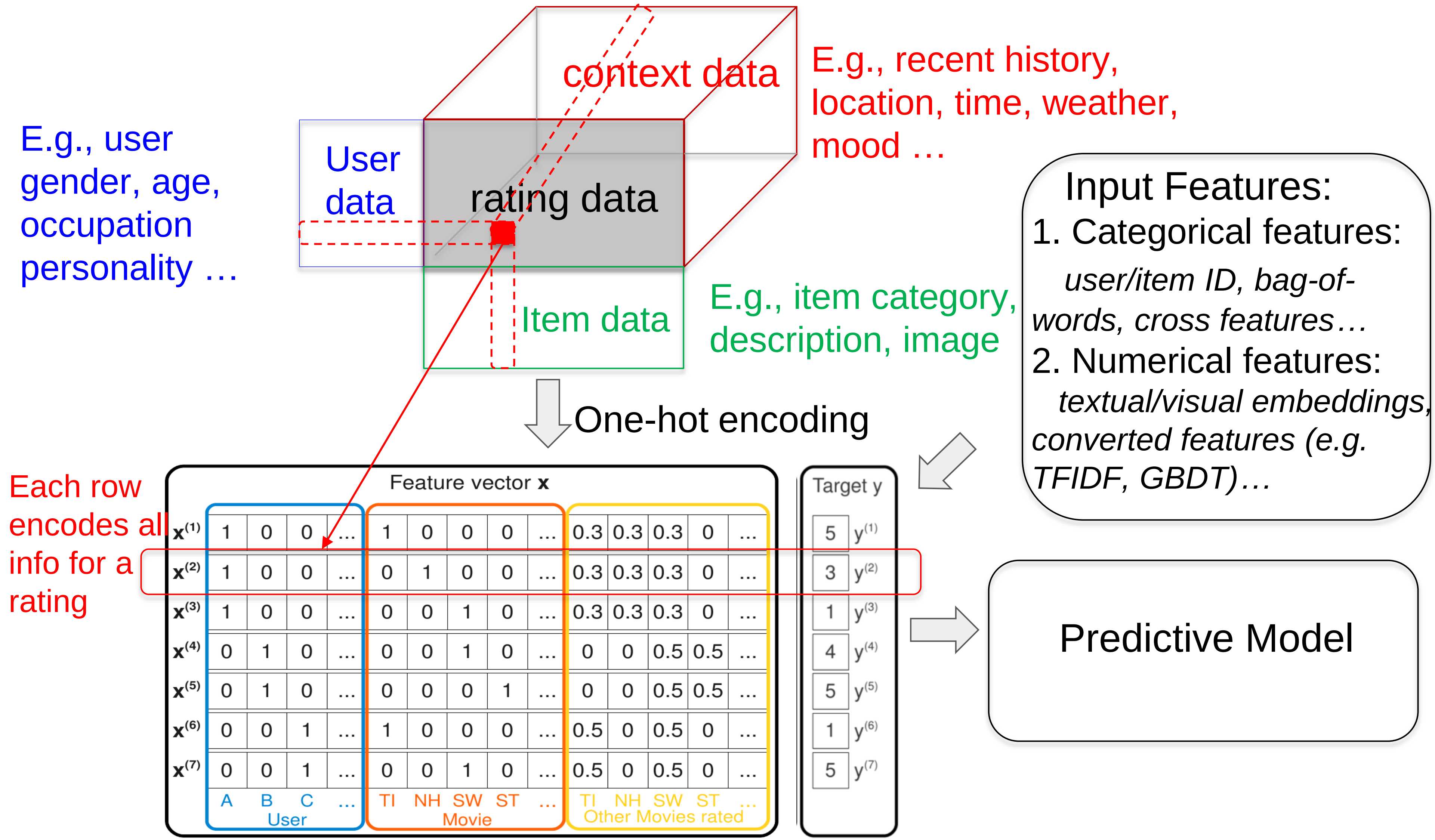
(Covington et al, Recsys'16)

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 …
- Powered by supervised learning methods. Optimization objective can be tailored for different scenarios, such as optimizing for:
  - Click-through rate (CTR)
  - Purchase
  - Impression

  ……

Input to Feature-based Models



	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...	Target y	
$\mathbf{x}^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	5	$y^{(1)}$
$\mathbf{x}^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	3	$y^{(2)}$
$\mathbf{x}^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	1	$y^{(3)}$
$\mathbf{x}^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	4	$y^{(4)}$
$\mathbf{x}^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	5	$y^{(5)}$
$\mathbf{x}^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	1	$y^{(6)}$
$\mathbf{x}^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	5	$y^{(7)}$

(Rendle, ICDM'10)

Key to Feature-based Models

- 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

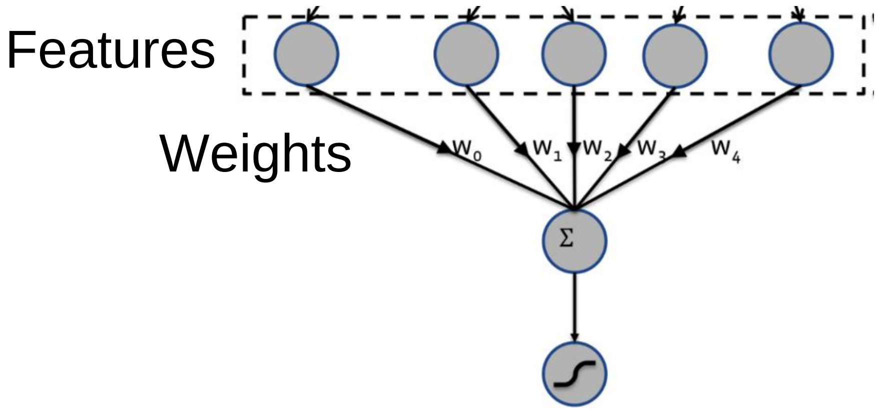
  E.g., users like to use food delivery apps at meal-time

  => 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

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

Pros:

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

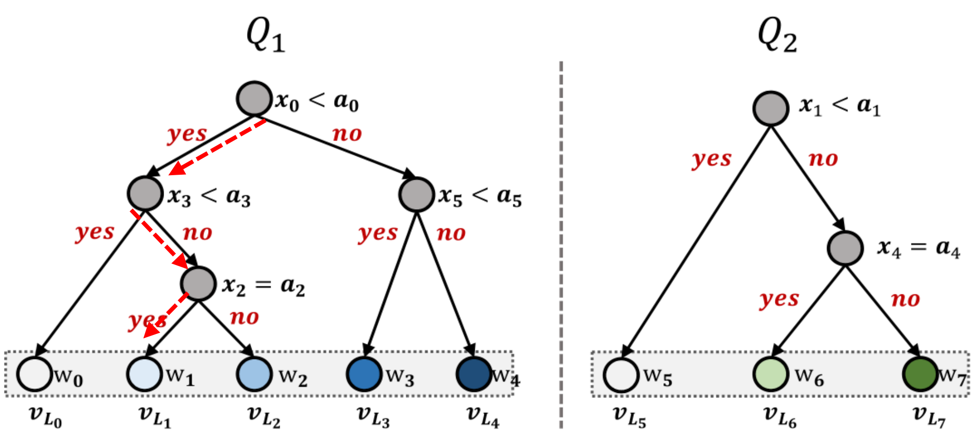
Cons:

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

Tree-based Models

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**.



leaf node v_{L_2} represents $[x_0 < a_0]\&[x_3 \geq a_3]\&[x_2 \neq a_2]$

Gradient Boosting Decision Trees (GBDT):

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

# of trees

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

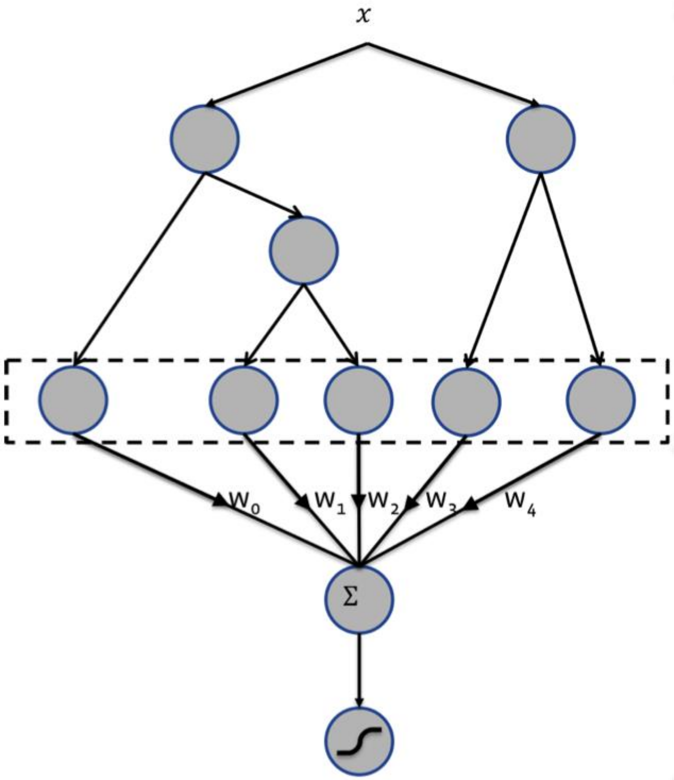
Prediction of the s-th tree

- Can capture more complex decision patterns than a single tree.

(Wang et al, WWW'18)

GBDT + LR

- 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.

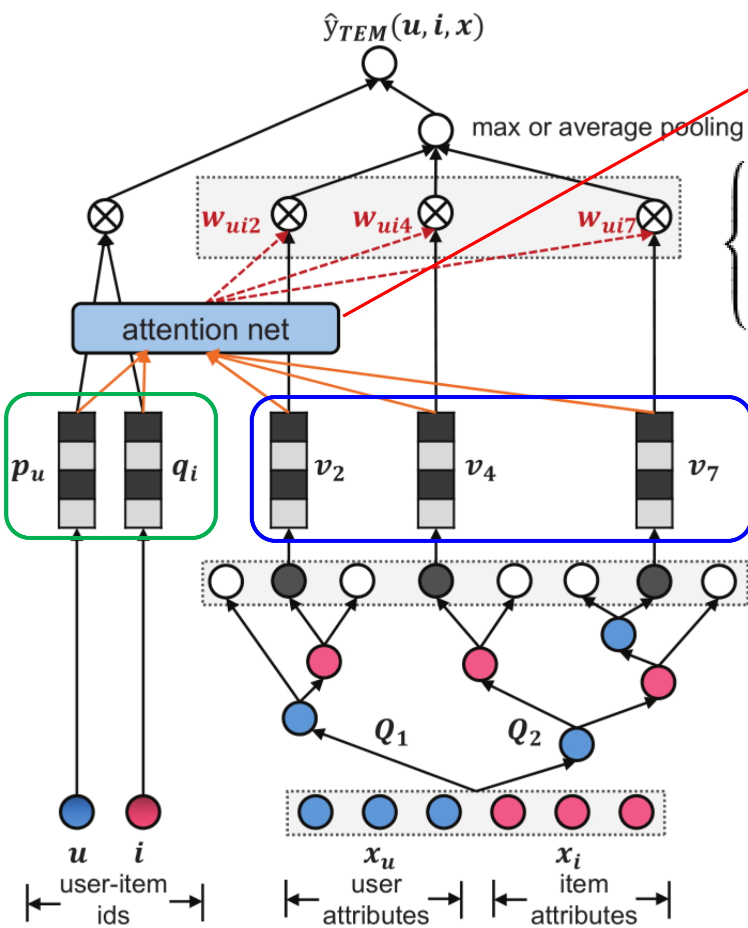
GBDT + Attention Net

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

Attention Net outputs weight of a cross feature for prediction (u, i) based on their embeddings:

$$\begin{cases} w'_{uil} = \mathbf{h}^{\top} ReLU(\mathbf{W}([\mathbf{p}_u \odot \mathbf{q}_i, \mathbf{v}_l]) + \mathbf{b}) \\ w_{uil} = \dfrac{exp(w'_{uil})}{\sum_{(u,i,\mathbf{x})\in O} exp(w'_{uil})} \end{cases}$$

Embeddings of user and item

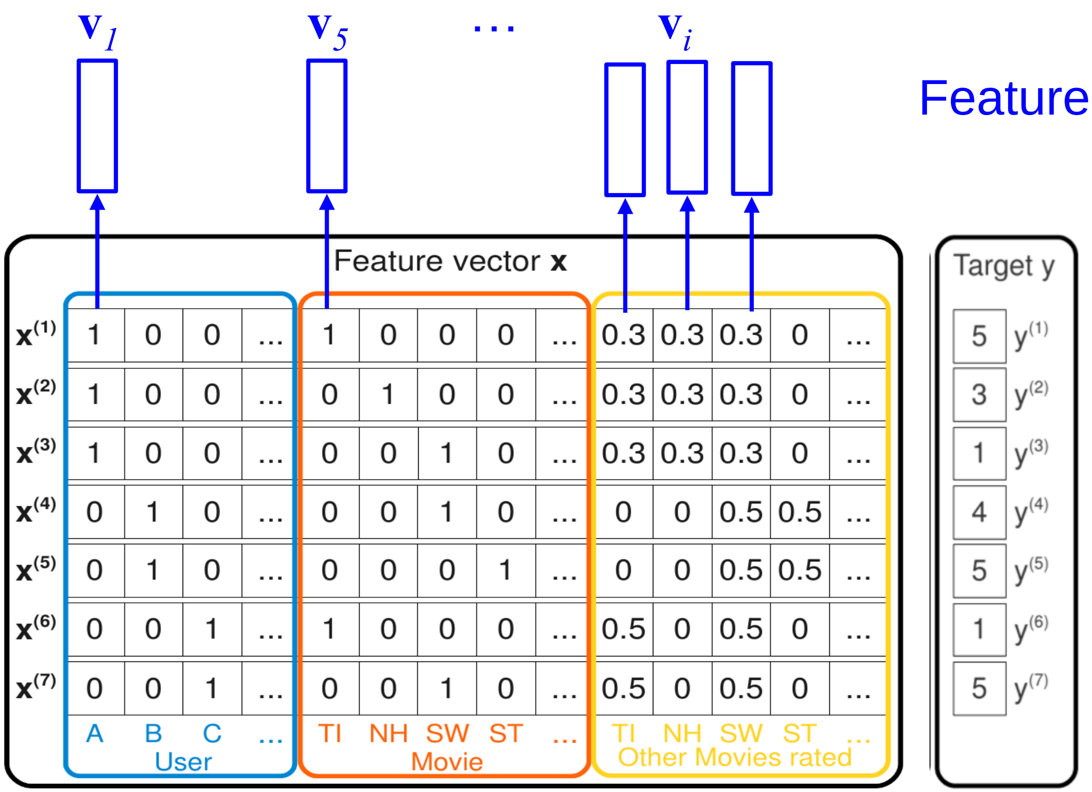


Embeddings of cross features

GBDT: generating cross features (each tree activates one leaf node)

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 (Rendle et al, ICDM'10)

- Extend LR by modeling pairwise interactions between feature embeddings with inner product:

Only nonzero features are considered

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

  - Note: self-interaction is not included: $\cancel{< \mathbf{v}_i, \mathbf{v}_i >}$.

- Example:

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

$y = w_{ESPN} + w_{Nike} + w_{Gender} + <\boldsymbol{v}_{ESPN}, \boldsymbol{v}_{Nike}> + <\boldsymbol{v}_{ESPN}, \boldsymbol{v}_{Male}> + <\boldsymbol{v}_{Nike}, \boldsymbol{v}_{Male}>$

Pros:

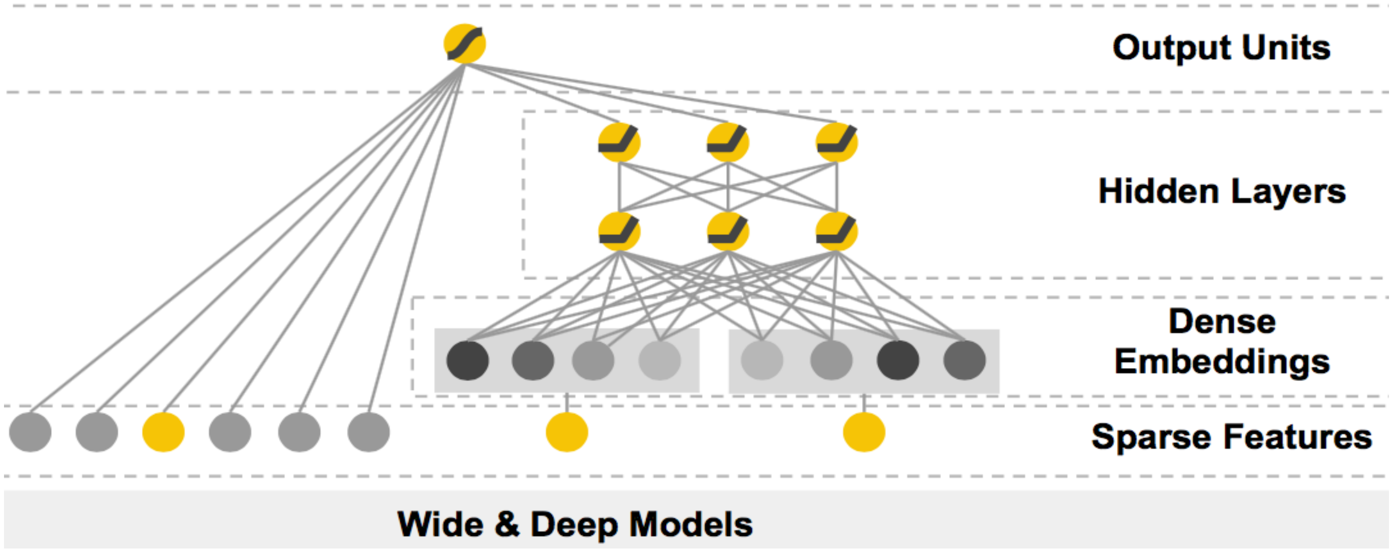
- Feature interactions are learned automatically.

Cons:

- Only 2nd-order feature interactions. (inefficient for higher order interactions)

Wide&Deep (Cheng et al, Recsys'16)

Google's App Recommender Ranking Solution in 2016



Standard MLP

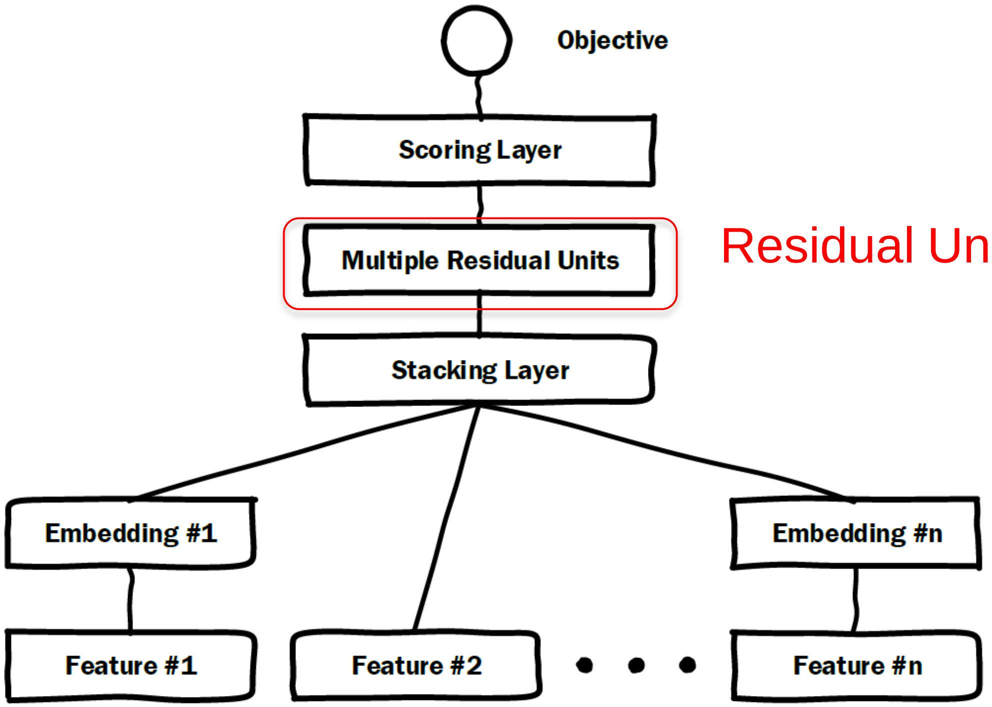
Embedding Concatenation

- The wide part is linear regression for memorizing seen feature interactions, which requires careful engineering on cross features. e.g., *AND(gender=female, 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.

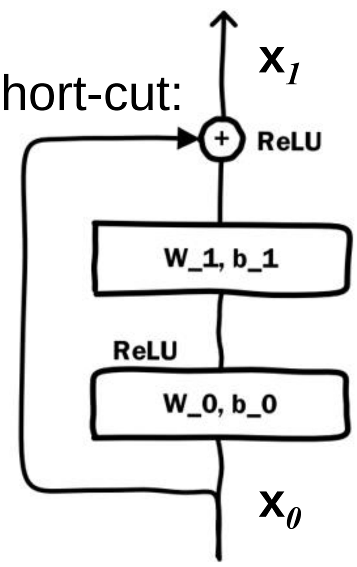
Deep Crossing (Shan et al, KDD'16)

Microsoft's CTR Prediction Solution in 2016:



Residual Unit:

short-cut:

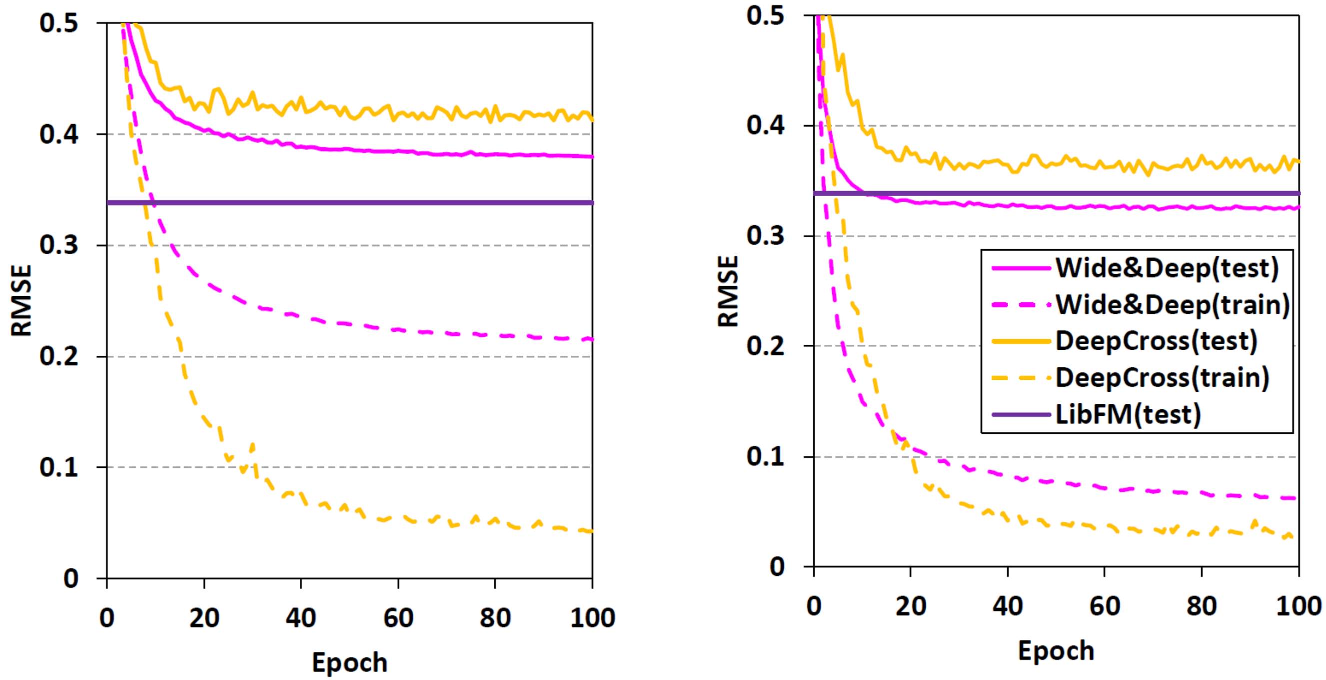


The main difference from Wide&Deep is the use of residual layers, which allow deeper network to be built (~10 layers).

Empirical Evidence (He and Chua, SIGIR'17)

- 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 (b) FM as pre-training

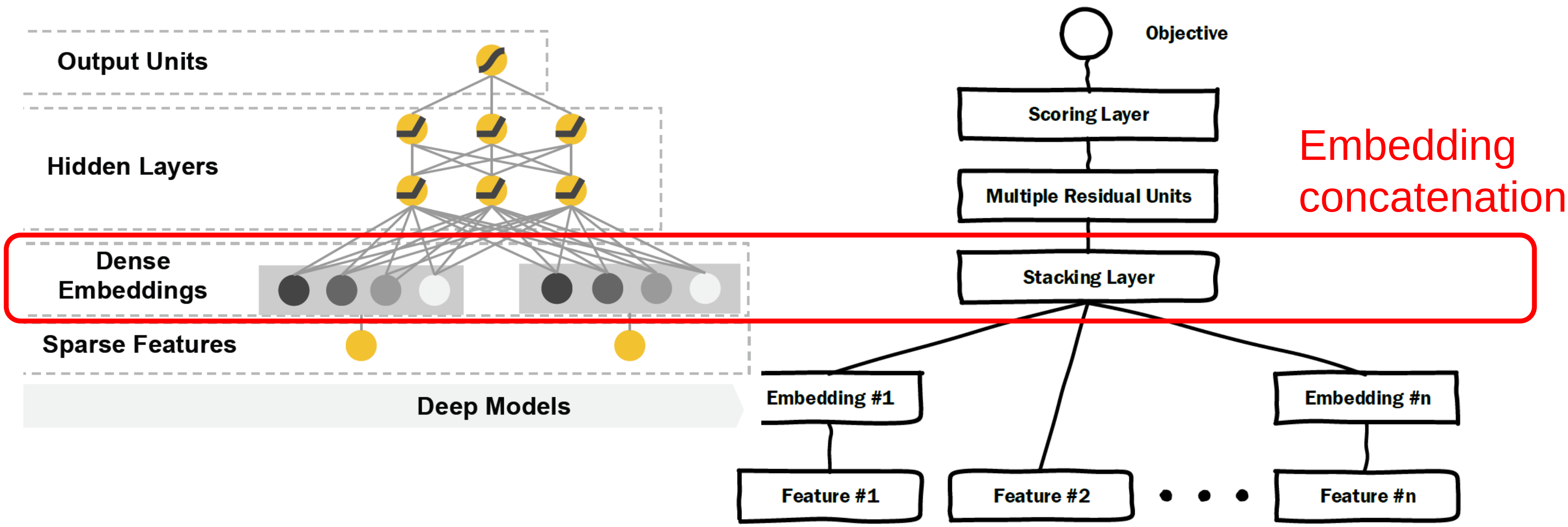
Some issues of DL methods:
Easy to overfit
Hard to train well
Need good init.

With random initialization, two deep methods underperform FM.

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

Why DNN is Ineffective?

Besides optimization difficulties, one reason is in model design:

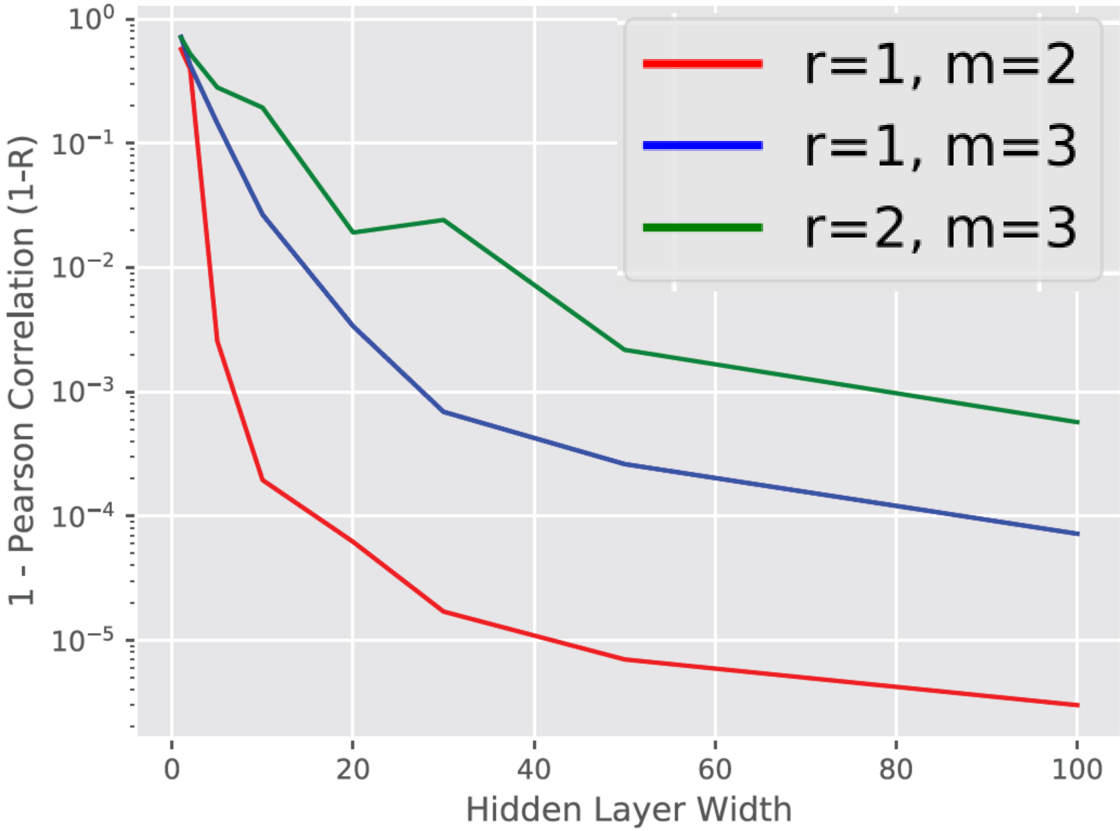


1. Embedding concatenation carries little information about feature interactions in the low level!
2. The structure of Concat+MLP is ineffective in learning the multiplicative relation (Beutel et al, WSDM’18).

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).



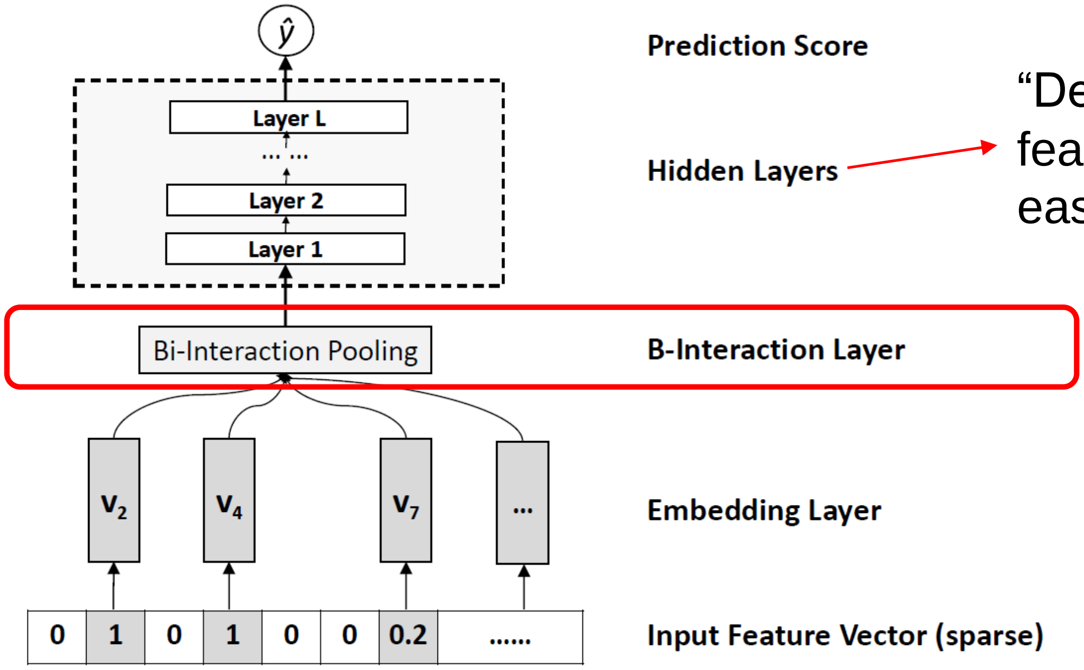
MLP can learn low-rank relation, but is inefficient in doing so!

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

Insight: need to augment DNN with multiplicative relation modeling

Neural Factorization Machine (He and Chua, SIGIR'17)

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



"Deep layers" learn **higher-order** feature interactions only, being much easier to train.

Bilinear Interaction Pooling:

$$f_{BI}(\mathcal{V}_x) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} x_i \mathbf{v}_i \odot x_j \mathbf{v}_j,$$

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

Empirical Evidence (He and Chua, SIGIR'17)

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

	Frappe		MovieLens	
Method	**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
Neural FM (1 layer)	**1.45M**	**0.3095**	**23.31M**	**0.4443**

1. Embedding methods learn interactions, better than simple linear models

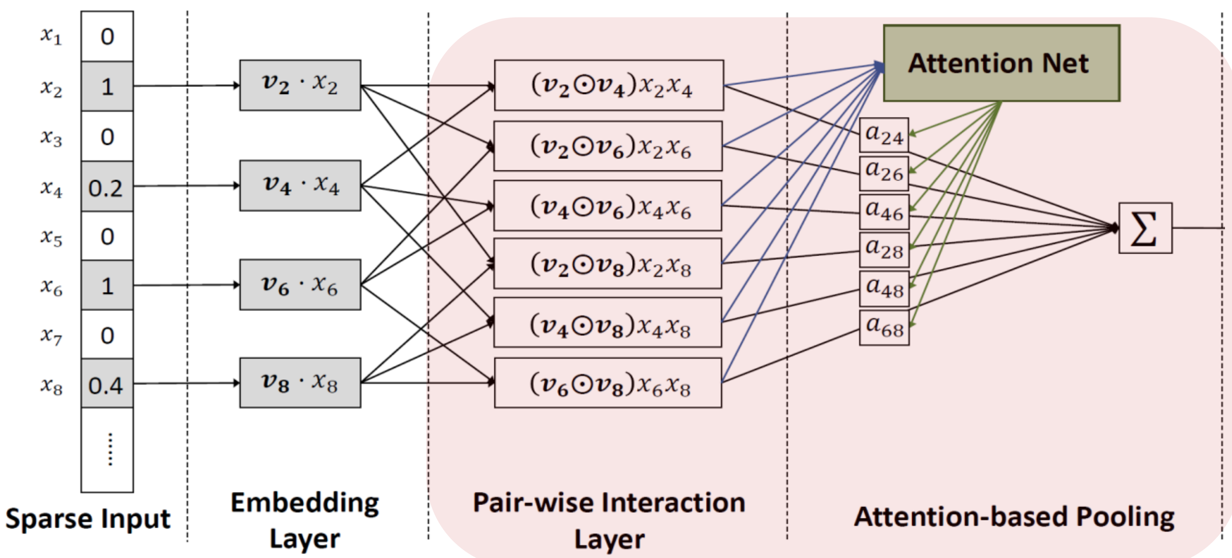
2. Deep embedding methods:
Wide&Deep = Concat+3 layers
DeepCross = Concat+10 layers

3. Neural FM
= BI pooling + 1 layer
Shallower but outperforming existing deeper methods with less parameters.

Codes: github.com/hexiangnan/neural_factorization_machine

Attentional FM (Xiao et al, IJCAI'17)

- 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.



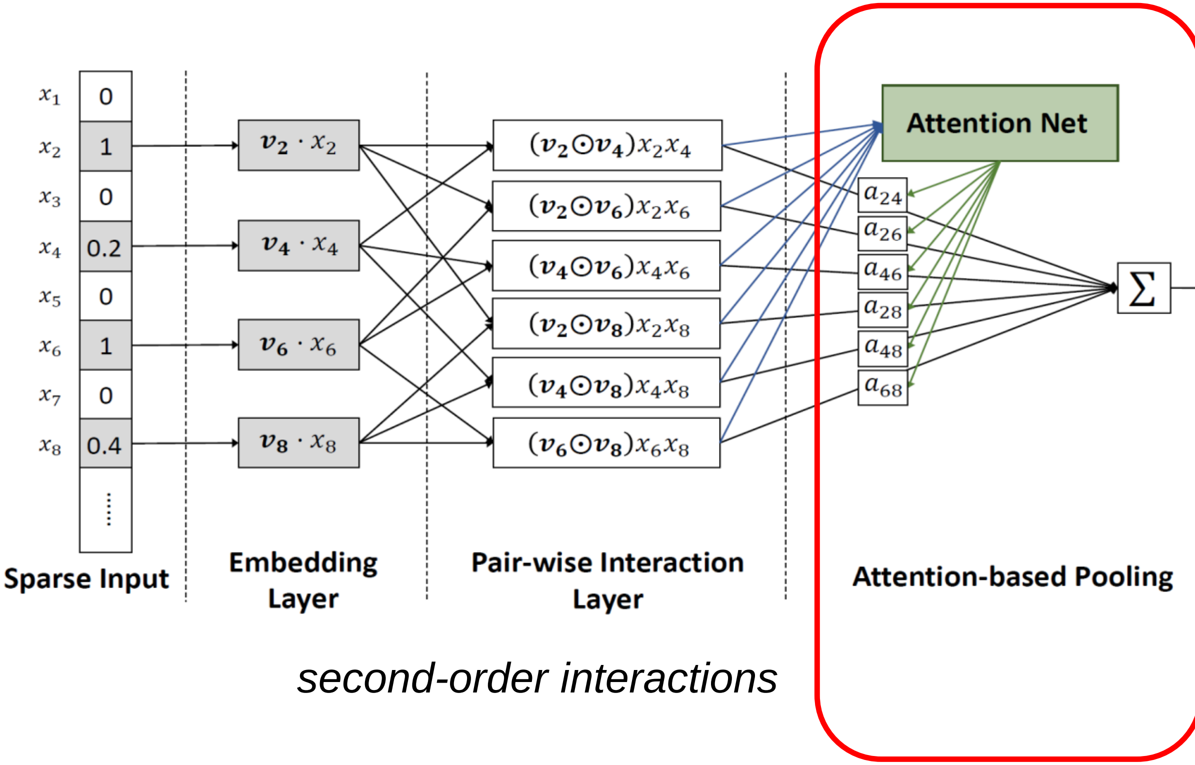
$$f_{ABI}(\mathcal{V}_x) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (x_i \mathbf{v}_i \odot x_j \mathbf{v}_j) a_{ij}$$

$$a'_{ij} = \mathbf{h}^T ReLU(\mathbf{W}(\mathbf{v}_i \odot \mathbf{v}_j) x_i x_j + \mathbf{b}),$$

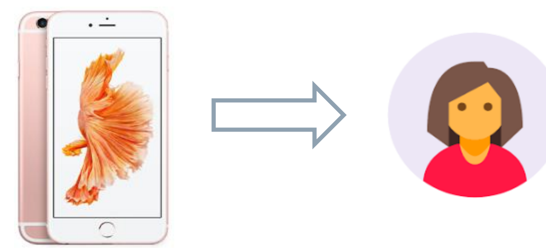
$$a_{ij} = \frac{\exp(a'_{ij})}{\sum_{(i,j)\in\mathcal{R}_x} \exp(a'_{ij})},$$

Explaining Recommendation with AFM

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 (Xiao et al, IJCAI'17)

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

	Frappe		MovieLens	
Method	**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
Neural FM (1 layer)	**1.45M**	**0.3095**	**23.31M**	**0.4443**
AFM (0 layer)	**1.45M**	**0.3102**	**23.26M**	**0.4325**

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

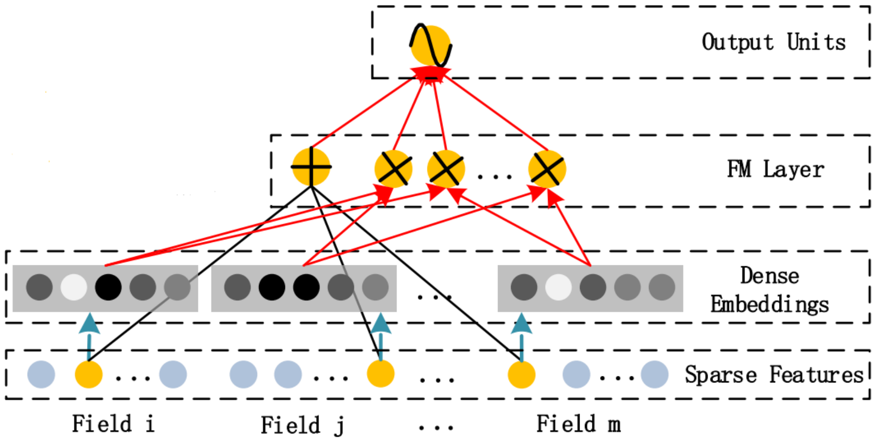
Adding hidden layers to AFM further improves.

Codes: github.com/hexiangnan/neural_factorization_machine

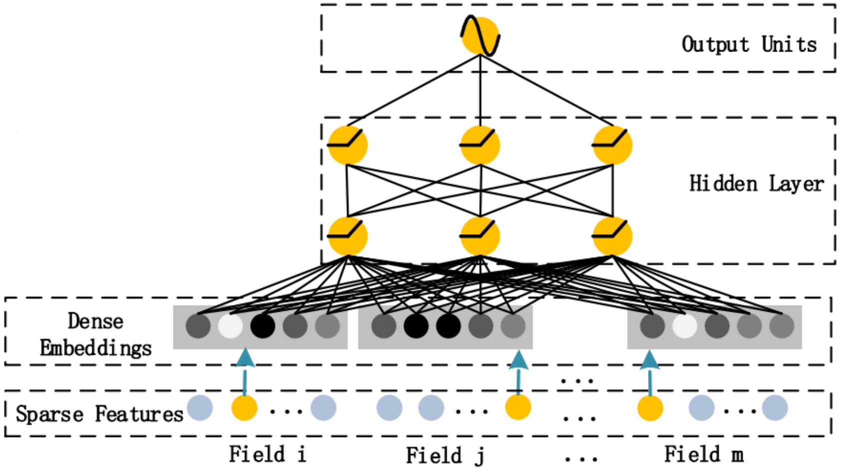
DeepFM (Guo et al., IJCAI'17)

- DeepFM ensembles FM and DNN and to learn both second-order and higher-order feature interactions:

FM structure:



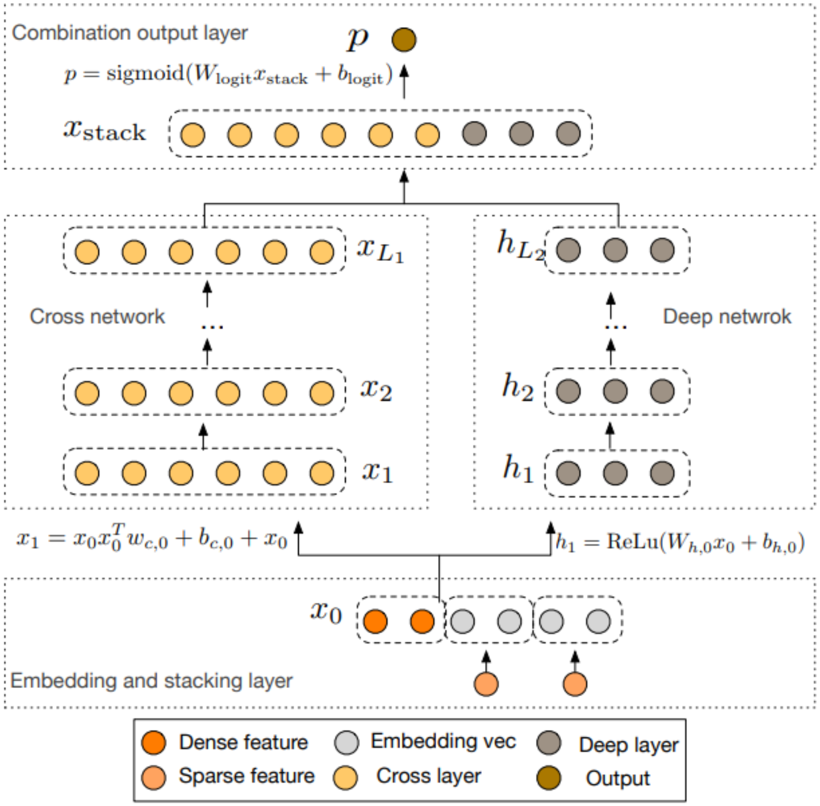
DNN structure:



Prediction Model: $\hat{y}_{DeepFM} = \hat{y}_{FM} + \hat{y}_{DNN}$

- Note: FM and DNN share the embedding layer.
- DeepFM learns DNN from the residual of FM
- NeuralFM learns DNN based on the latent space of FM

Deep & Cross (Wang et al., ADKDD'17)



- Cross layer

$$\mathbf{x}_{l+1} = \mathbf{x}_0 \mathbf{x}_l^T \mathbf{w}_l + \mathbf{b}_l + \mathbf{x}_l = f(\mathbf{x}_l, \mathbf{w}_l, \mathbf{b}_l) + \mathbf{x}_l$$

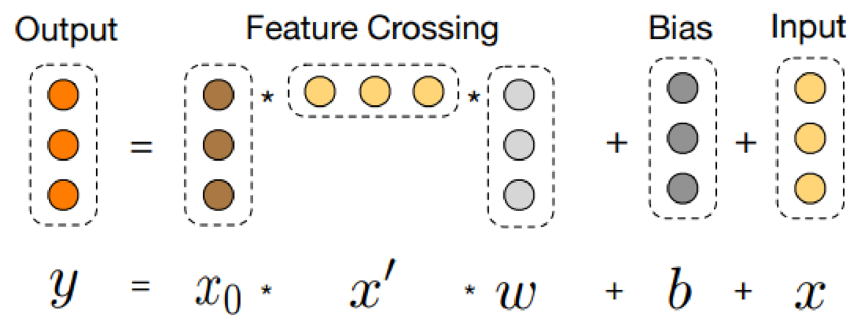


Figure 2: Visualization of a cross layer.

Simple yet effective: stacking more layers explicitly models higher-order interactions

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.

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.

Reference

- 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.

Reference

- 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.