Model-Based Collaborative Filtering

Recommender Systems

Dimitris Sacharidis

model-based collaborative filtering

- Matrix Factorization
- **Factorization Machines**
- **Sparse Linear Method (SLIM)**
- **Neural Networks**
- **Handling Implicit Feedback**
    - **Weighted Matrix Factorization (WMF)**
    - **Bayesian Personalized Ranking (BPR)**

factorization machines

view of factorization machines

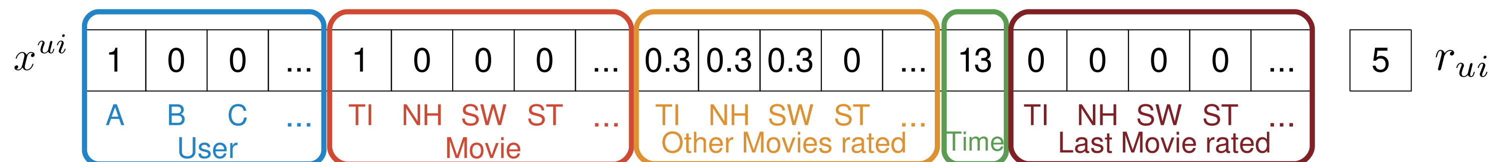
- can be regarding as a generalization of matrix factorization that can
  - encode **multiple attributes (**sources of information) about users and items
  - consider **higher-degree** dependencies between features
- a training **example** (rating) can include attributes about **which user** rated **which item**, **when**, etc.

example x^{ui} *encoding multiple sources of information about user* u *and item* i

User				Movie					Other Movies rated					Time	Last Movie rated					rating r_{ui}
A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...	Time	TI	NH	SW	ST	...	
1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5
1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3
1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1
0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4
0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5
0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1
0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5

[2012 TIST S. Rendle] *Factorization Machines with libFM*

building up to factorization machines



how to **predict ratings** given all these attributes?

- **basic linear regression**
  - learn the appropriate weight w_p for each attribute p

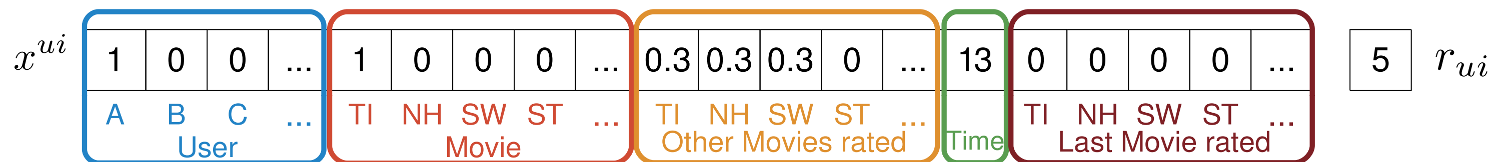
$$\hat{r}_{ui} = \mu + \sum_p w_p x_p^{ui}$$

μ is the intercept/bias = global average rating

- **linear regression with higher-order terms**
  - introduce additional attributes capturing interactions (up to degree 2) between attributes
  - also learn the weights $w_{p,p'}$ for each pair of attributes p, p'

$$\hat{r}_{ui} = \mu + \sum_p w_p x_p^{ui} + \sum_p \sum_{p' \neq p} w_{pp'} x_p^{ui} x_{p'}^{ui}$$

building up to factorization machines



what is the problem with the higher-order linear regression idea?

$$\hat{r}_{ui} = \mu + \sum_p w_p x_p^{ui} + \sum_p \sum_{p' \neq p} w_{pp'} x_p^{ui} x_{p'}^{ui}$$

too many weights $w_{p,p'}$ to learn:

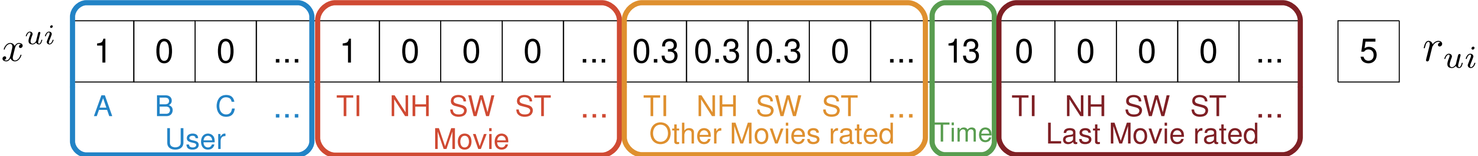
e.g., for m users and n items in the order of $(n+m)^2$

solution: **factorize** the weight matrix consisting of all $w_{p,p'}$

- each attribute p has a **latent vector** v_p
- weights are now computed by the dot product $w_{pp'} = v_p^\intercal v_{p'}$

[2012 TIST S. Rendle] *Factorization Machines with libFM*

factorization machines



$$\hat{r}_{ui} = \mu + \sum_{p} w_p x_p^{ui} + \sum_{p} \sum_{p' \neq p} w_{pp'} x_p^{ui} x_{p'}^{ui} \quad \text{where} \quad w_{pp'} = v_p^\intercal v_{p'}$$

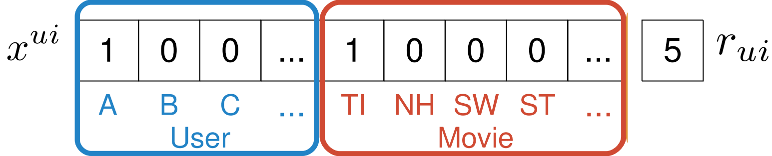
parameters to learn?

- for each feature p
  - a weight term w_p
  - a latent vector v_p of dimensionality k

[2012 TIST S. Rendle] *Factorization Machines with libFM*

factorization machine vs. matrix factorization

- if only user and item attributes are kept



- **factorization machine = matrix factorization** with baseline estimate

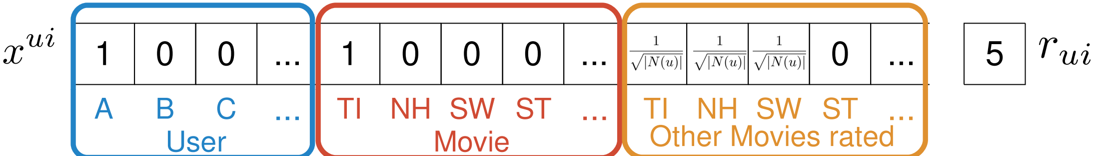
0-order | 1-order | 2-order

$$\hat{r}_{ui} = \mu + \sum_p w_p x_p^{ui} + \sum_p \sum_{p' \neq p} w_{pp'} x_p^{ui} x_{p'}^{ui} \quad \text{where} \quad w_{pp'} = v_p^\intercal v_{p'}$$

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^\intercal q_i$$

factorization machine vs. matrix factorization

- if only **user**, **item**, and **other movies rated** attributes are kept



- **factorization machine** looks like **SVD++**

0-order, 1-order, 2-order

$$\hat{r}_{ui} = \mu + \sum_{p} w_p x_p^{ui} + \sum_{p} \sum_{p' \neq p} w_{pp'} x_p^{ui} x_{p'}^{ui}$$

where $w_{pp'} = v_p^{\intercal} v_{p'}$

$$\hat{r}_{ui} = \mu + b_u + b_i + \cdots + q_i^{\intercal} \left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right) + \cdots$$

baseline estimate

SVD++

factorization machines

(+) can easily include additional sources of information

just like matrix factorization

(-) are a bit **rigid**: they always consider all pairwise interactions between attributes

they cannot exactly reproduce SVD++

sparse linear methods (SLIM)

motivation

- recall the prediction formula for I-I collaborative filtering:

$$\hat{r}_{ui} = \overline{r_i} + \frac{\sum_{j \in N(i;u)} w_{ij}(r_{uj} - \overline{r_j})}{\sum_{j \in N(i;u)} |w_{ij}|}$$

- this is a weighted average
    - how are the weights computed?

- what if the weights were learned by a model?

SLIM

- first, simplify the formula
    - consider all items, not only the neighbors
    - ignore deviations, directly compute rating prediction
    - assume weights are normalized

$$\hat{r}_{ui} = \sum_j w_{ij} r_{uj}$$

- or in matrix form:

$$\hat{R} = RW$$

[2011 ICDM X. Ning, G. Karypis] *SLIM: Sparse Linear Methods for Top-N Recommender Systems*

SLIM

$$\hat{R} = RW$$

m users, *n* items

$\hat{R}$ = R · W

$m \times n$ $\quad m \times n$ $\quad n \times n$

- goal: find a sparse (L1-regularized) matrix *W*
  - why sparse?
- are users modeled in SLIM?
- how to recommend to a new user?

factorized SLIM

- many parameters in the model $\hat{R} = RW$
  - how many?
- not enough user-item interactions to learn them

- solution: compute a factorization of weight matrix $W = Q^{\intercal} Y$
  - where Q, Y are $k \times n$

- a weight is computed as the inner product

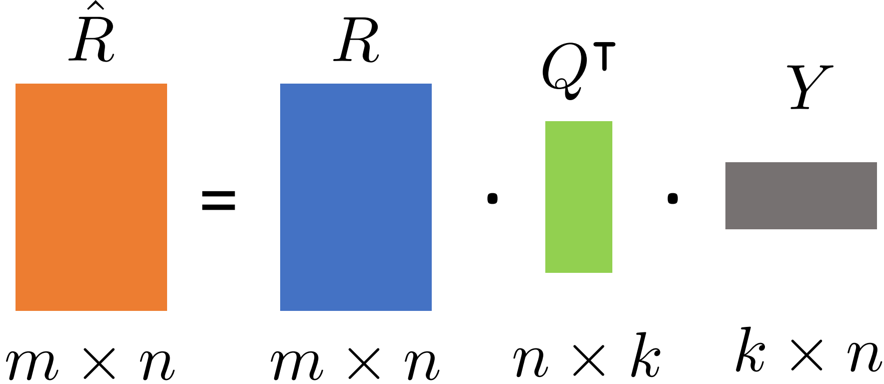
$$w_{ij} = q_i^{\intercal} y_j$$

[2013 KDD S. Kabbur et al.] *FISM: Factored Item Similarity Models for Top-N Recommender Systems*

factorized SLIM

- let's see this in matrix form

$$\hat{R} = RQ^\intercal Y$$



- so, the model predicts a rating as:

$$\hat{r}_{ui} = q_i^\intercal \sum_{j \in I_u} r_{uj} \cdot y_j$$

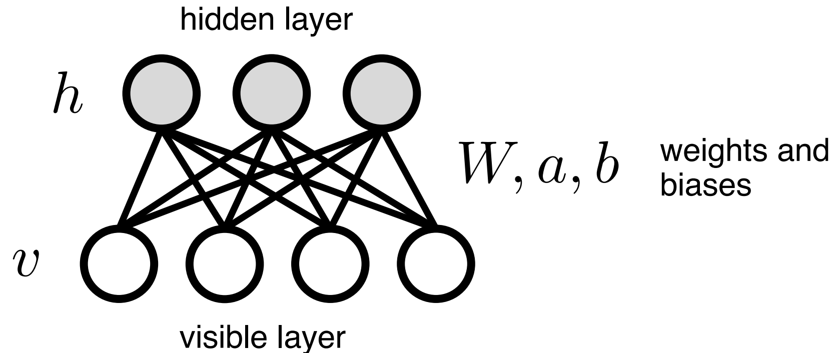
item model *user model*

- have you seen something like this before? in SVD++

neural network approaches

restricted Boltzmann machines

- the first neural net approach appeared on 2007 based on G. Hinton's **Restricted Boltzmann Machines** (RBMs, stochastic neural nets used for **generative** learning):
  - one visible (input/output) layer
  - one hidden layer with **stochastic binary** units (0/1 with some prob.)
  - weights, biases **encode** the input into a binary representation
  - a binary encoding is **decoded** to an output via the weights
  - weights are learned by **contrastive divergence** (RBM learning)

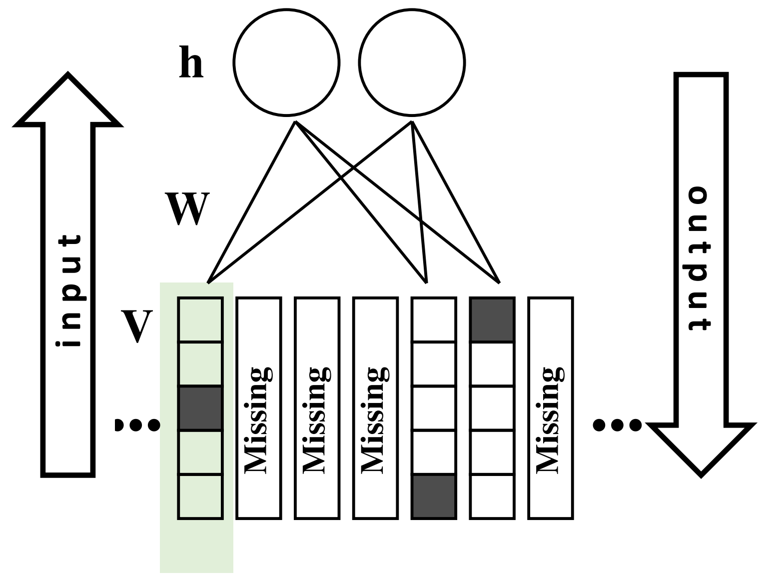


restricted Boltzmann machines

- **user-based input**: ratings for each user
  - ratings represented as a vector of length 5, where entries represent a star
  - number of input units is number of items
- **output** are the probabilities for star ratings

issues:

- can only handle integer ratings
- computing predictions can be expensive
- **obsolete** – no longer relevant

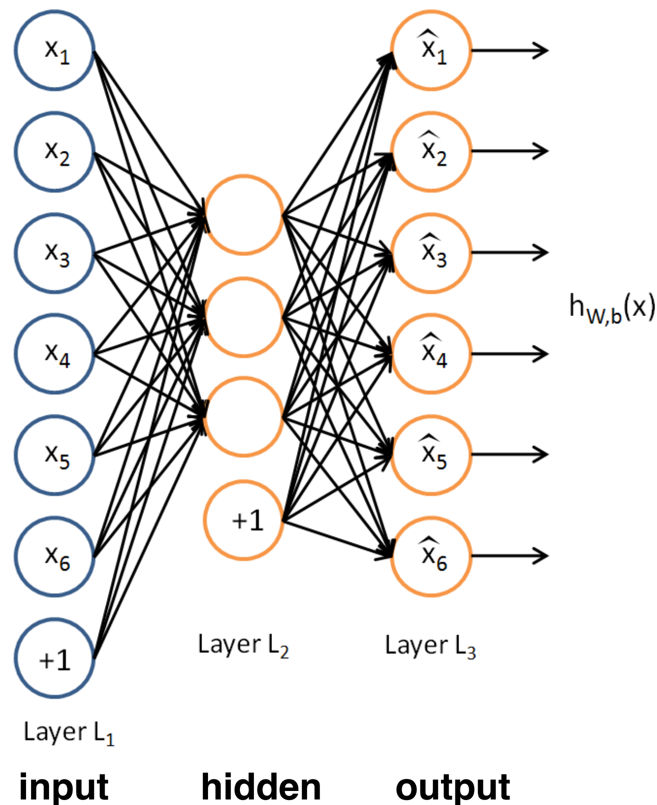


the rating of an item: each box represents a star
input: filled box is the given rating
output: boxes contain probabilities

[2007 ICML R. Salakhutdinov et al.] *Restricted Boltzmann Machines for Collaborative Filtering*

autoencoders

- **autoencoders** are (shallow) feed forward neural nets used for **unsupervised learning** and trained with backpropagation (SGD)
- **hidden** layer contains much fewer units than **input** layer
  - **encodes** the input, i.e.,
  - **embeds** it into a lower dimensionality space
- **output** layer has as many units as **input**
  - **decodes** the encoding/embedding
- **training goal** is to make output **match** input (+ other regularization objectives)



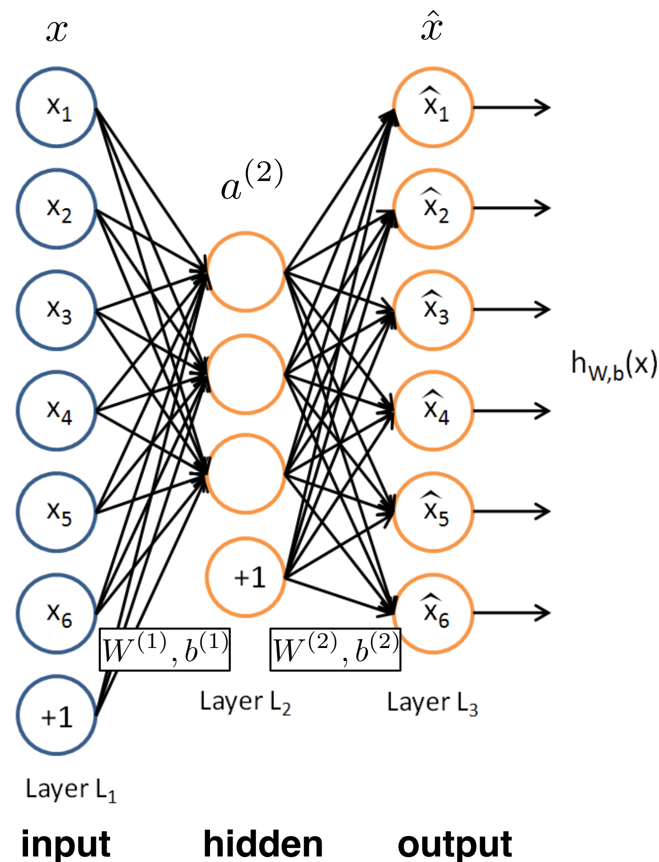
[CS294A Lecture Notes 2001, Sparse Autoencoders, Andrew Ng]

autoencoders

- parameters:
  - weights $W = \{W^{(1)}, W^{(2)}\}$
  - biases $b = \{b^{(1)}, b^{(2)}\}$

- what they compute:
  - at layer 2: $a^{(2)} = f(W^{(1)}x + b^{(1)})$
  - at layer 3: $\hat{x} \equiv h_{W,b}(x) = f(W^{(2)}a^{(2)} + b^{(2)})$
  - using some nonlinear *activation function* (sigmoid, tanh, relu) f

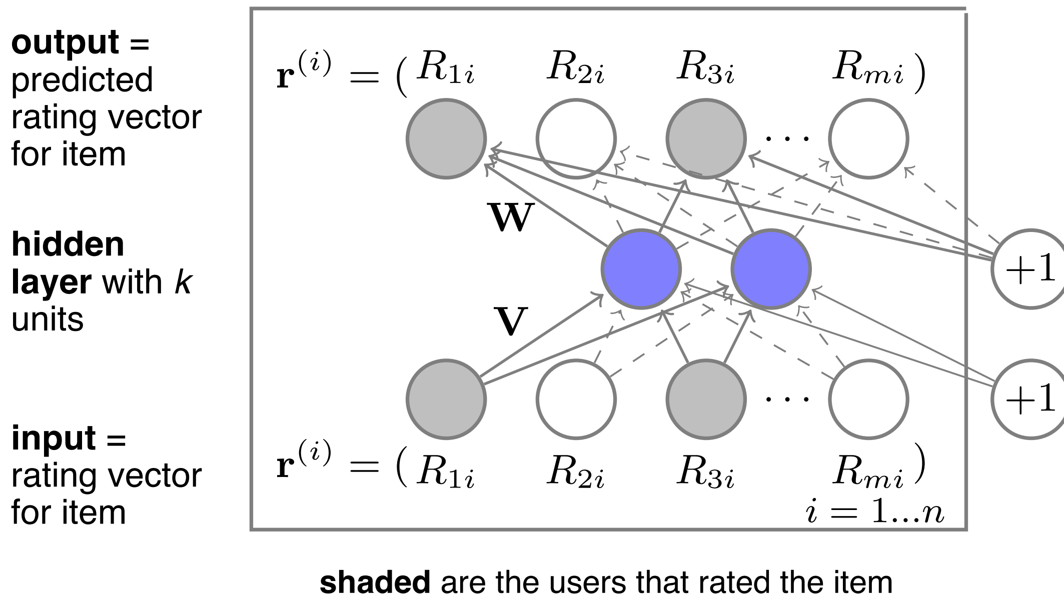
- goal: $\hat{x} \equiv h_{W,b}(x) \approx x$



[CS294A Lecture Notes 2001, Sparse Autoencoders, Andrew Ng]

autoencoders

- AutoRec uses an autoencoder on ratings for each item (**item-based input**) shown below
    - can also operate on ratings for each user (**user-based input**)

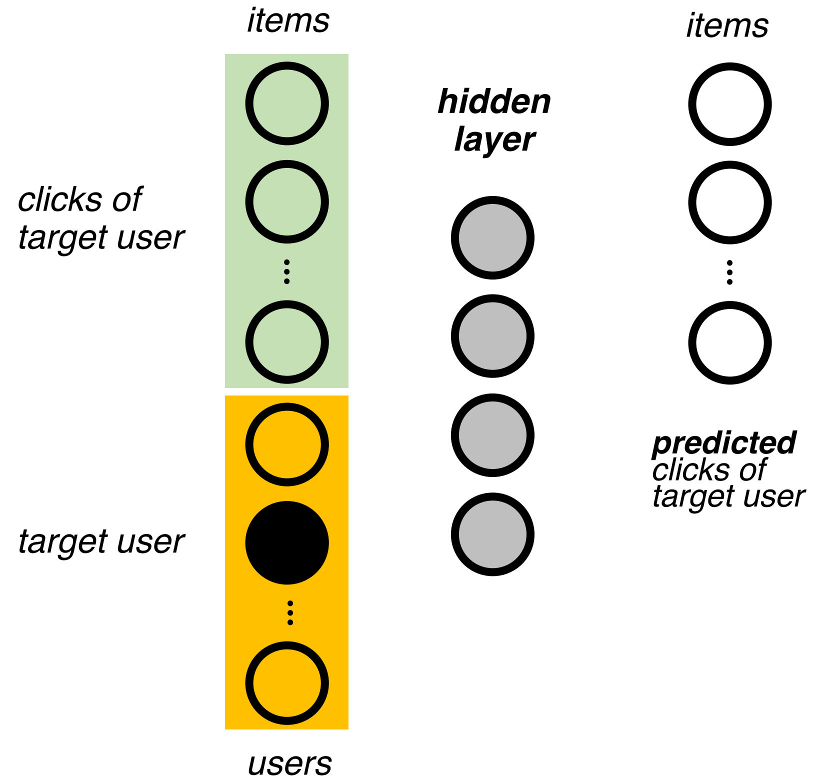


shaded are the users that rated the item

[2015 WWW S. Sedhain et al.] *AutoRec: Autoencoders Meet Collaborative Filtering*

denoising autoencoders

- CDAE use denoising autoencoders for **implicit feedback** datasets
  - **denoising autoencoders** are like autoencoders but learn over **corrupted** (by noise) input
- CDAE accepts **user-based inputs** plus a one-hot encoding of the target user
  - each example contains the **clicks** of the target user
- **hidden** layer of *k* units
- **output** is the prediction of clicks of the target user



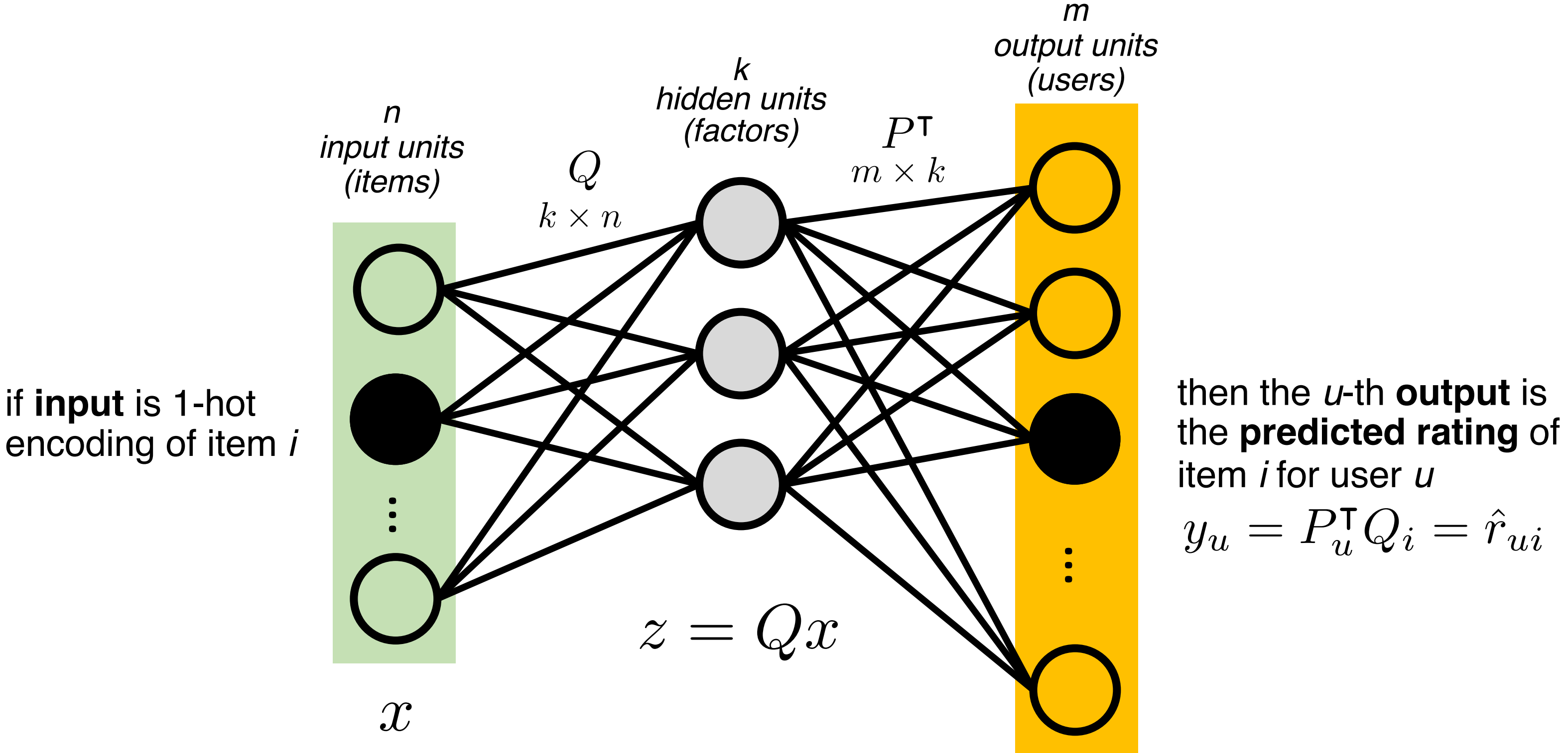
[2016 WSDM Y. Wu et al.] Collaborative *Denoising Auto-encoders for Top-N Recommender Systems*

neural networks

- neural-network based approaches are more generic than matrix factorization

- question: how can you implement **matrix factorization** as a **neural network**?

matrix factorization as a neural network



- ignore biases
- assume linear activations

Handling Implicit Feedback

the challenge with implicit feedback data

- assume **binary** implicit feedback; ratings matrix contains only 1s
- how do you **train** a matrix factorization model?

- option 1. **Ignore missing ratings**; just as standard MF
  - what does the model learn if all it sees are 1s?
  - it will learn to predict just 1s!

- option 2. **Treat missing ratings as 0s** (negative feedback); learn from all ratings
  - a better option, but what would a **perfect** model learn?
    - predict 1 to known ratings and 0 to missing ratings
    - not what we want

the challenge with implicit feedback data

- option 2 is the only way to go
- we just need to avoid learning to predict just 0s for the missing ratings and 1s for the observed ratings
  - some **small changes** to the matrix factorization idea
- two different methods:
  - weighted matrix factorization
  - Bayesian personalized ranking

Weighted Matrix Factorization (WMF)

confidence in ratings

- how confident can we be that a **1** is a **positive** feedback?
  - not much; e.g., the user might **buy** an item and:
    - **not like** it -> **negative**
    - **like** it -> **positive**

- similarly, how confident are we that **0** is a **negative** feedback?
  - not much; e.g., the user might **not buy** an item because:
    - they are **not aware** of it -> could be **positive** or **negative**
    - they are **aware** of it and chose not to buy it -> **negative**

confidence in ratings

- better questions to pose

- when can we **increase our confidence** that a 1 is a **positive** feedback?
  - if we have non-binary feedback; e.g., more plays of a song means likely more positive
  - if we have other external sources of information

- when can we **increase our confidence** that 0 is a **negative** feedback?
  - if we know that the item was recommended/viewed but was not clicked
  - if we assume that popular items are well-known
  - if we have other external sources of information

- in general, we assume that we have a **confidence weight** w_{ui} in the observed or missing rating of item i by user u

possible definitions of confidence weights

for **observed** feedback

- let c_{ui} denote the **count** feedback associated with that user-item pair (rating r_{ui} is still binary)
  - e.g., c_{ui} is the number of views/plays etc.
- then we can set $w_{ui} = 1 + \alpha \cdot c_{ui}$
- higher count → more confidence in **positive** feedback

for **missing** feedback

- let f_i denote the **popularity** of item i
- then we can set $w_{ui} = w \cdot \frac{f_i^a}{\sum_j f_j^a}$
  - where w is a fixed term; exponent α controls the significance of popularity
- higher popularity → more confidence in **negative** feedback

[2008 ICDM Y. Hu et al.] *Collaborative Filtering for Implicit Feedback Datasets*

[2016 SIGIR X. He et al.] *Matrix Factorization for Online Recommendation with Implicit Feedback*

weighted matrix factorization

- prediction formula remains the same (ignore bias terms):

$$\hat{r}_{ui} = q_i^\intercal p_u$$

- so we still need to learn parameters in P, Q
- but the cost becomes:

$$J = \frac{1}{n \cdot m} \sum_{u,i} w_{ui}\, e_{ui}^2 + \lambda(\|P\|^2 + \|Q\|^2)$$

sum over all user-item pairs

confidence weights

- with two changes
  - confidence weights
  - the sum is over observed (1s) and missing (0s) feedback
    - not only the observed as in standard MF

weighted matrix factorization

$$J = \frac{1}{n \cdot m} \sum_{u,i} w_{ui}\, e_{ui}^2 + \lambda(\|P\|^2 + \|Q\|^2)$$

- how do the weights affect the learning?
  - when confidence is high, we care a lot about the error
  - when confidence is low, we are more tolerant to errors
- to learn the user and item factors
  - one can use SGD
  - or an efficient version of ALS

Bayesian personalized ranking (BPR)

consider pairs of items

- assumption: an item (implicitly) rated by a user is **preferred** over a non-rated item

- in other words, for any pair of items i, j:
  - if both are rated, then **no preference** is expressed
  - if both are not rated, then **no preference** is expressed
  - if only one is rated, then one is **preferred** over the other: e.g., $i > j$

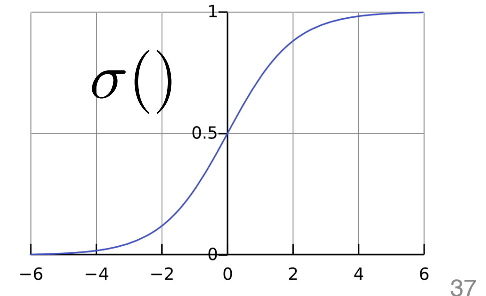
learning over pairs of items

- training examples are triples (u, i, j) where $i > j$
- the goal is to learn a function that predicts 1 for such triples

- specifically, we want to learn a score $\hat{x}_{uij}$
- such that when constrained to [0,1] represents the **preference probability** that $i > j$

$$\sigma(\hat{x}_{uij}) = P(i > j)$$

  - where $\sigma()$ is the logistic function that returns values in [0,1]



learning over pairs of items

- so score $\hat{x}_{uij}$ captures preference of *i* over *j* for user *u*
- but how to compute it?
- using some model (e.g., matrix factorization) to predict the rating of user *u* to the two items i, j

$$\hat{x}_{uij} = \hat{r}_{ui} - \hat{r}_{uj}$$

- item i is preferred over j, i.e., $\hat{r}_{ui} > \hat{r}_{uj}$
  - difference is positive
  - the bigger the difference the closer to 1 the probability $P(i > j)$ becomes

- item j is preferred over i, i.e., $\hat{r}_{uj} > \hat{r}_{ui}$
  - difference is negative
  - the bigger the (absolute) difference the closer to 0 the probability $P(i > j)$ becomes

[2009 UAI S. Rendle et al.] *BPR: Bayesian Personalized Ranking from Implicit Feedback*

learning over pairs of items

- how many training examples are there?
  - for each rated item i there are **too many unrated items** j
  - $O(n^2)$ possible positive-negative (rated-unrated) (i, j) pairs per user
- solution: reduce the number of pairs examined
  - for each rated item **sample** a few negative (unrated) items
- in the end:
  - learn over the created (u, i, j) training examples
  - the parameters of the underlying model (matrix factorization)

[2009 UAI S. Rendle et al.] *BPR: Bayesian Personalized Ranking from Implicit Feedback*

point-wise vs. pair-wise learning

- **point-wise** learning (e.g., WMF)
  - focus on making accurate rating predictions
  - i.e., optimizes *prediction* accuracy
- **pair-wise** learning (e.g., BPR)
  - focus on distinguishing the relative order (preference) between two items
  - i.e., optimizes *ranking* accuracy
    - more suitable for top-N recommendations