Learning to Recommend via Matrix Factorization/Completion

Piyush Rai

Introduction to Machine Learning (CS771A)

October 30, 2018

Recommendation Systems

- The goal is to recommend more relevant items to users, based on previous interactions
- Used by many services
- Note: Relevance is subjective here
- A notion of relevance: I will buy/watch/like items that are similar to ones I did in the past
- Has been a very active research topic (in ML and allied areas) for a long time
  - Even a dedicated conference focusing on this topic specifically - RecSys

Recommendation Systems as Matrix Completion

- One of the most popular ways to solve the RecSys problem
- Suppose we have this partially complete ratings matrix

?	?	3	?	4	?
?	?	?	3	?	?
?	2	?	?	?	4
3	?	?	?	3	?
?	?	4	?	?	?
4	4	?	?	?	3

- Once completed, the completed matrix can be used to recommend “best” items for a given user
  - For example: Recommend the items that have a high (predicted) rating for the user
- Note: In addition to the user-item matrix, we may have additional info about the user/items
  - Some examples: User meta-data, item content description, user-user network, item-item similarity, etc.

Some Notation

?	?	3	?	4	?
?	?	?	3	?	?
?	2	?	?	?	4
3	?	?	?	3	?
?	?	4	?	?	?
4	4	?	?	?	3

- Let's denote the user-item $N \times M$ ratings matrix as $\mathbf{X}$ (many entries missing)
- Suppose $\Omega = \{(n, m)\}$ is the set of indices for observed ratings
- Suppose Ω_{r_n} is the set of indices of items already rated by user n
- Suppose Ω_{c_m} is the set of indices of users who already rated item m

A Simple Heuristic: Item based "Collaborative Filtering"

?	?	3	?	4	?
?	?	?	3	?	?
?	2	?	?	?	4
3	?	?	?	3	?
?	?	4	?	?	?
4	4	?	?	?	3

- For each user-item pair (n, m), compute the missing rating X_{nm} as

$$X_{nm} \approx \frac{1}{|\Omega_{r_n}|} \sum_{m' \in \Omega_{r_n}} S^{(I)}_{mm'} X_{nm'}$$

where $S^{(I)}_{mm'} \in (0, 1)$ is the similarity between items m and m' (suppose known)

A Simple Heuristic: User based "Collaborative Filtering"

?	?	3	?	4	?
?	?	?	3	?	?
?	2	?	?	?	4
3	?	?	?	3	?
?	?	4	?	?	?
4	4	?	?	?	3

- For each user-item pair (n, m), compute the missing rating X_{nm} as

$$X_{nm} \approx \frac{1}{|\Omega_{c_m}|} \sum_{n' \in \Omega_{c_m}} S_{nn'}^{(U)} X_{n'm}$$

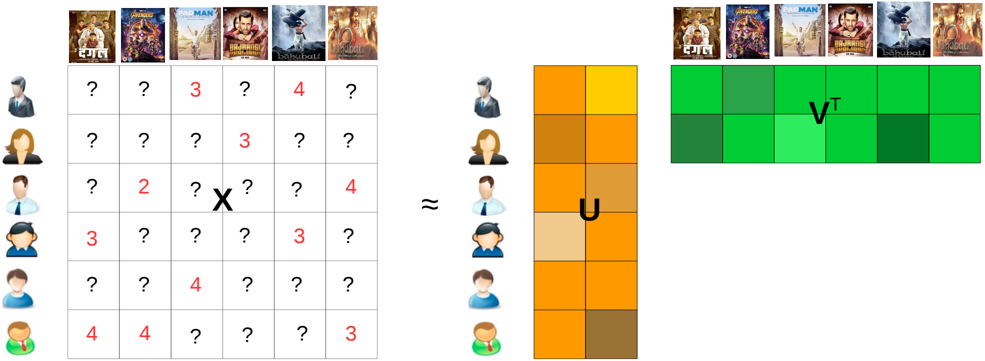
where $S_{nn'}^{(U)} \in (0, 1)$ is the similarity between users n and n' (suppose known)

Limitations of Item/User Based Approach

?	?	3	?	4	?
?	?	?	3	?	?
?	2	?	?	?	4
3	?	?	?	3	?
?	?	4	?	?	?
4	4	?	?	?	3

- User-User or Item-Item similarities may not be known beforehand
- We may have very little data in the user-item matrix and averaging may not be reliable

Towards a better approach: Matrix Factorization

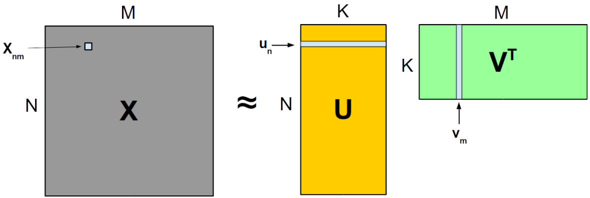


If we can do the above factorization then any missing $X_{nm} \approx \boldsymbol{u}_n^\top \boldsymbol{v}_m$

Matrix Factorization

- Given a matrix $\mathbf{X}$ of size $N \times M$, approximate it as a product of two matrices

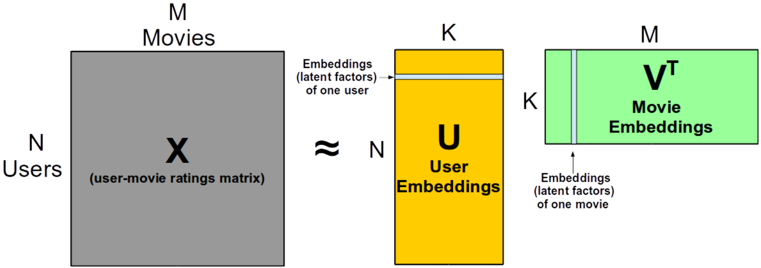
$$\mathbf{X} \approx \mathbf{U}\mathbf{V}^\top$$



- $\mathbf{U}$: $N \times K$ latent factor matrix
  - Each row of $\mathbf{U}$ represents a K-dim latent factor $\boldsymbol{u}_n$
- $\mathbf{V}$: $M \times K$ latent factor matrix
  - Each row of $\mathbf{V}$ represents a K-dim latent factor $\boldsymbol{v}_n$
- Each entry of $\mathbf{X}$ can be written as: $X_{nm} \approx \boldsymbol{u}_n^\top \boldsymbol{v}_m = \sum_{k=1}^{K} u_{nk} v_{mk}$

Why Matrix Factorization?

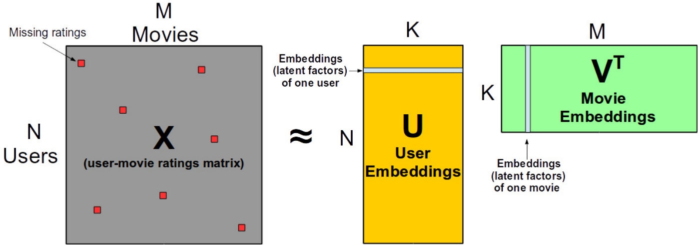
- The latent factors can be used/interpreted as "embeddings" or "learned features"



- Especially useful for learning good features for "dyadic" or relational data
  - Examples: Users-Movies ratings, Users-Products purchases, etc.
- If $K \ll \min\{M, N\} \Rightarrow$ then can also be seen as dimensionality reduction or a "low-rank factorization" of the matrix $\mathbf{X}$ (somewhat like SVD)

Why Matrix Factorization?

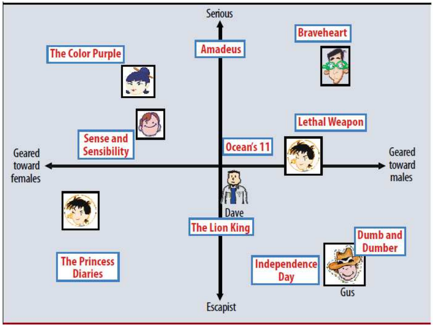
- Can also predict the missing/unknown entries in the original matrix



- Yes. **U** and **V** can be learned even when the matrix **X** is only partially observed (we'll see shortly)
- After learning **U** and **V**, any missing X_{nm} can be approximated by $\boldsymbol{u}_n^\top \boldsymbol{v}_m$
- $\mathbf{UV}^\top$ is the best low-rank matrix that approximates the full **X**
- Note: The "Netflix Challenge" was won by a matrix factorization method

Interpreting the Embeddings/Latent Factors

- Embeddings/latent factors can often be interpreted. E.g., as "genres" if $\mathbf{X}$ represents a user-movie rating matrix. A cartoon with $K = 2$ shown below

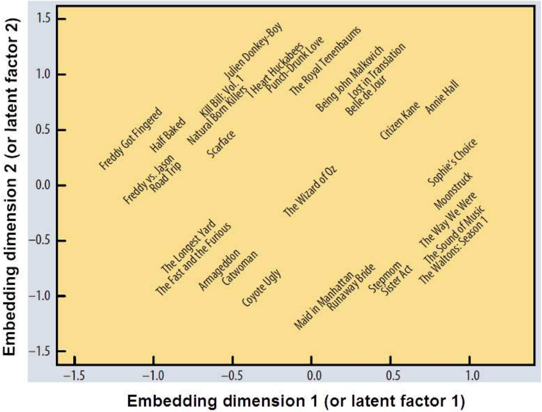


- Similar things (users/movies) get embedded nearby in the embedding space (two things will be deemed similar if their embeddings are similar). Thus useful for computing similarities and/or making recommendations

Picture courtesy: Matrix Factorization Techniques for Recommender Systems: Koren et al, 2009

Interpreting the Embeddings/Latent Factors

- Another illustation of 2-D embeddings of the movies only



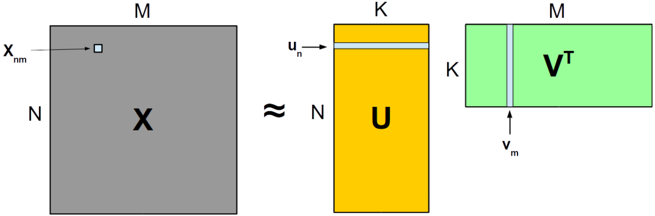
- Similar movies will be embedded at nearby locations in the embedding space

Picture courtesy: Matrix Factorization Techniques for Recommender Systems: Koren et al, 2009

Solving Matrix Factorization

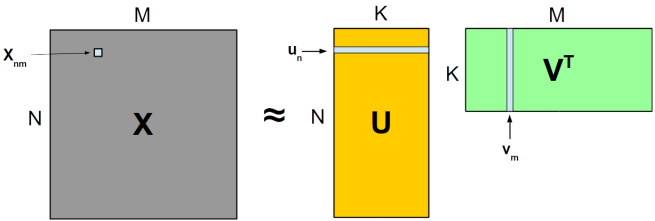
Matrix Factorization

- Recall our matrix factorization model: $\mathbf{X} \approx \mathbf{U}\mathbf{V}^\top$
- Goal: learn $\mathbf{U}$ and $\mathbf{V}$, given a subset Ω of entries in $\mathbf{X}$ (let's call it $\mathbf{X}_\Omega$)
- Recall our notations:
  - $\Omega = \{(n, m)\}$: X_{nm} is observed
  - Ω_{r_n}: column indices of observed entries in row n of $\mathbf{X}$
  - Ω_{c_m}: row indices of observed entries in column m of $\mathbf{X}$



Matrix Factorization

- We want $\mathbf{X}$ to be as close to $\mathbf{UV}^\top$ as possible



- Let's define a squared "loss function" over the observed entries in $\mathbf{X}$

$$\mathcal{L} = \sum_{(n,m)\in\Omega} (X_{nm} - \boldsymbol{u}_n^\top \boldsymbol{v}_m)^2$$

- Here the latent factors $\{\boldsymbol{u}_n\}_{n=1}^N$ and $\{\boldsymbol{v}_m\}_{m=1}^M$ are the unknown parameters
- Squared loss chosen only for simplicity; other loss functions can be used
- How do we learn $\{\boldsymbol{u}_n\}_{n=1}^N$ and $\{\boldsymbol{v}_m\}_{m=1}^M$?

Alternating Optimization

- We will use an ℓ_2 regularized version of the squared loss function

$$\mathcal{L} = \sum_{(n,m)\in\Omega} (X_{nm} - \boldsymbol{u}_n^\top \boldsymbol{v}_m)^2 + \sum_{n=1}^{N} \lambda_U ||\boldsymbol{u}_n||^2 + \sum_{m=1}^{M} \lambda_V ||\boldsymbol{v}_m||^2$$

- A **non-convex** problem. Difficult to optimize w.r.t. $\boldsymbol{u}_n$ and $\boldsymbol{v}_m$ jointly.
- One way is to solve for $\boldsymbol{u}_n$ and $\boldsymbol{v}_m$ in an alternating fashion, e.g.,
  - $\forall n$, fix all variables except $\boldsymbol{u}_n$ and solve the optim. problem w.r.t. $\boldsymbol{u}_n$

$$\arg\min_{\boldsymbol{u}_n} \sum_{m\in\Omega_{r_n}} (X_{nm} - \boldsymbol{u}_n^\top \boldsymbol{v}_m)^2 + \lambda_U ||\boldsymbol{u}_n||^2$$

  - $\forall m$, fix all variables except $\boldsymbol{v}_m$ and solve the optim. problem w.r.t. $\boldsymbol{v}_m$

$$\arg\min_{\boldsymbol{v}_m} \sum_{n\in\Omega_{c_m}} (X_{nm} - \boldsymbol{u}_n^\top \boldsymbol{v}_m)^2 + \lambda_V ||\boldsymbol{v}_m||^2$$

  - Iterate until not converged
- Each of these subproblems has a simple, convex objective function
- Convergence properties of such methods have been studied extensively

The Solutions

- Easy to show that the problem

$$\arg\min_{\boldsymbol{u}_n} \sum_{m\in\Omega_{r_n}} (X_{nm} - \boldsymbol{u}_n^\top \boldsymbol{v}_m)^2 + \lambda_U ||\boldsymbol{u}_n||^2$$

.. has the solution

$$\boldsymbol{u}_n = \left(\sum_{m\in\Omega_{r_n}} \boldsymbol{v}_m \boldsymbol{v}_m^\top + \lambda_U \mathbf{I}_K\right)^{-1} \left(\sum_{m\in\Omega_{r_n}} X_{nm}\boldsymbol{v}_m\right)$$

- Likewise, the problem

$$\arg\min_{\boldsymbol{v}_m} \sum_{n\in\Omega_{c_m}} (X_{nm} - \boldsymbol{u}_n^\top \boldsymbol{v}_m)^2 + \lambda_V ||\boldsymbol{v}_m||^2$$

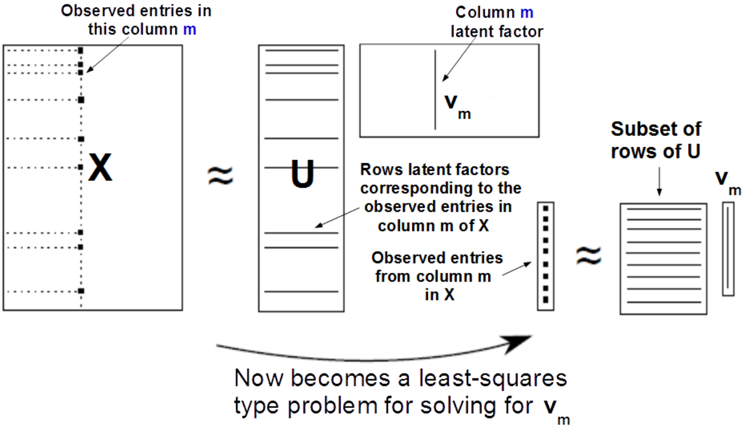
.. has the solution

$$\boldsymbol{v}_m = \left(\sum_{n\in\Omega_{c_m}} \boldsymbol{u}_n \boldsymbol{u}_n^\top + \lambda_V \mathbf{I}_K\right)^{-1} \left(\sum_{n\in\Omega_{c_m}} X_{nm}\boldsymbol{u}_n\right)$$

- Note that this is very similar to (regularized) least squares regression
- Thus matrix factorization can be also seen as a sequence of regression problems (one for each latent factor)

Matrix Factorization as Regression

Suppose we are solving for $\boldsymbol{v}_m$ (with $\mathbf{U}$ and all other $\boldsymbol{v}_m$'s fixed)



Can think of solving for $\boldsymbol{u}_n$ (with $\mathbf{V}$ and all other $\boldsymbol{u}_n$'s fixed) in the same way

Matrix Factorization as Regression

- A very useful way to understand matrix factorization
- Can modify the regularized least-squares like objective

$$\arg\min_{\boldsymbol{u}_n} \sum_{m \in \Omega_{r_n}} (X_{nm} - \boldsymbol{u}_n^\top \boldsymbol{v}_m)^2 + \lambda_U \boldsymbol{u}_n^\top \boldsymbol{u}_n$$

.. using other loss functions and regularizers

- Some possible modifications:
  - If entries in the matrix $\mathbf{X}$ are binary, counts, etc. then we can replace the squared loss function by some other loss function (e.g., logistic or Poisson)
  - Can impose other constraints on the latent factors, e.g., non-negativity, sparsity, etc. (by changing the regularizer)
  - Can think of this also as a probabilistic model (a likelihood function on X_{nm} and priors on the latent factors $\boldsymbol{u}_n$, $\boldsymbol{v}_m$) and do MLE/MAP

Matrix Factorization: The Complete Algorithm

- Input: Partially complete matrix $\mathbf{X}_\Omega$
- Initialize the latent factors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_M$ randomly
- Iterate until not converged
  - Update each row latent factor $\boldsymbol{u}_n$, $n = 1, \ldots, N$ (can be in parallel)

$$\boldsymbol{u}_n = \left(\sum_{m \in \Omega_{r_n}} \boldsymbol{v}_m \boldsymbol{v}_m^\top + \lambda_U \mathbf{I}_K \right)^{-1} \left(\sum_{m \in \Omega_{r_n}} X_{nm} \boldsymbol{v}_m \right)$$

  - Update each column latent factor $\boldsymbol{v}_m$, $m = 1, \ldots, M$ (can be in parallel)

$$\boldsymbol{v}_m = \left(\sum_{n \in \Omega_{c_m}} \boldsymbol{u}_n \boldsymbol{u}_n^\top + \lambda_V \mathbf{I}_K \right)^{-1} \left(\sum_{n \in \Omega_{c_m}} X_{nm} \boldsymbol{u}_n \right)$$

- Final prediction for any (missing) entry: $X_{nm} = \boldsymbol{u}_n^\top \boldsymbol{v}_m$

A Faster Algorithm via SGD

- Alternating optimization is nice but can be slow (note that it requires matrix inversion with cost $O(K^3)$ for updating each latent factor $\boldsymbol{u}_n, \boldsymbol{v}_m$)
- An alternative is to use stochastic gradient descent (SGD). In each round, select a randomly chosen entry X_{nm} with $(n, m) \in \Omega$
- Consider updating $\boldsymbol{u}_n$. For loss function $\sum_{m \in \Omega_{r_n}} (X_{nm} - \boldsymbol{u}_n^\top \boldsymbol{v}_m)^2 + \lambda_U ||\boldsymbol{u}_n||^2$, the stochastic gradient w.r.t. $\boldsymbol{u}_n$ using this randomly chosen entry X_{nm} is

$$-(X_{nm} - \boldsymbol{u}_n^\top \boldsymbol{v}_m)\boldsymbol{v}_m + \lambda_U \boldsymbol{u}_n$$

- Thus the SGD update for $\boldsymbol{u}_n$ will be

$$\boldsymbol{u}_n = \boldsymbol{u}_n - \eta(\lambda_U \boldsymbol{u}_n - (X_{nm} - \boldsymbol{u}_n^\top \boldsymbol{v}_m)\boldsymbol{v}_m)$$

- Likewise, the SGD update for $\boldsymbol{v}_m$ will be

$$\boldsymbol{v}_m = \boldsymbol{v}_m - \eta(\lambda_V \boldsymbol{v}_m - (X_{nm} - \boldsymbol{u}_n^\top \boldsymbol{v}_m)\boldsymbol{u}_n)$$

- The SGD algorithm chooses a random entry X_{nm} in each iteration, updates $\boldsymbol{u}_n, \boldsymbol{v}_m$, and repeats until convergece ($\boldsymbol{u}_n$'s,$\boldsymbol{v}_m$'s randomly initialized).

Explicit Feedback vs Implicit Feedback Data

- Often the user-item matrix $\mathbf{X}$ is a binary matrix
- $X_{nm} = 1$ means user n watched and liked on the item m
- What does $X_{nm} = 0$ mean? Watched but didn't like?
- Or $X_{nm} = 0$ mean user wasn't explosed to this item?
- There is no way to distinguish such 0s in $\mathbf{X}$
- Such binary $\mathbf{X}$ is called "implicit feedback" as opposed to explicit feedback (e.g., ratings)
- Such data needs more careful modeling (other loss functions, not squared/logistic)

0	0	1	0	1	0
0	0	0	1	0	0
0	1	0	0	0	1
1	0	0	0	1	0
0	0	1	0	0	0
1	1	0	0	0	1

- Some popular schemes include
  - Downweightling the contribution of 0s in the loss function
  - Use ranking based loss function, e.g., want $\boldsymbol{u}_n^\top \boldsymbol{v}_m > \boldsymbol{u}_n^\top \boldsymbol{v}_{m'}$ if $X_{nm} = 1$ and $X_{nm'} = 0$

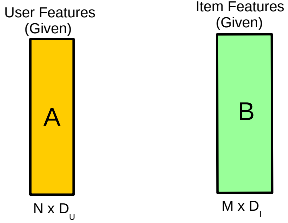
Inductive Matrix Completion

- "Inductive" here means that we would like to extrapolate to new users/items
- Also known as the "cold-start" problem in RecSys literature

- The matrix factorization approach would need latent factors for the new users/items
- How to compute these latent factors without any ratings for such users/items?

Inductive Matrix Completion

- Often we have some additional "meta-data" about the users or items (or both)
  - Example: User profile info, item description/image, etc.
- Can use this meta-data to get some features
- Assume we have D_u features for each user and D_I features for each item

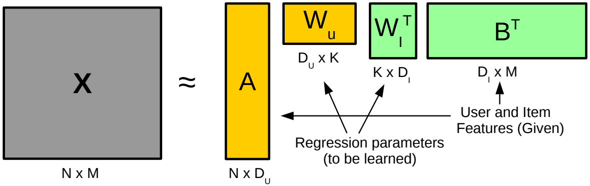


- One possibility now: Use these features/meta-data to get the latent factors for users/items

Matrix Factorization for Inductive Matrix Completion

- Basic idea: Assume $\mathbf{X} \approx \mathbf{U}\mathbf{V}^\top$ but regress $\mathbf{U}$ and $\mathbf{V}$ using $\mathbf{A}$ and $\mathbf{B}$, respectively

$$\mathbf{U} = \mathbf{A}\mathbf{W}_u \quad \text{and} \quad \mathbf{V} = \mathbf{B}\mathbf{W}_I$$



- The loss function will be

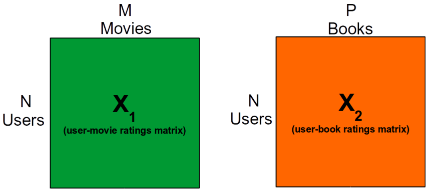
$$||\mathbf{X} - \mathbf{U}\mathbf{V}^\top||^2 = ||\mathbf{X} - (\mathbf{A}\mathbf{W}_U) \times (\mathbf{B}\mathbf{W}_I)^\top||^2$$

- We optimize this loss function w.r.t. $\mathbf{W}_U$ and $\mathbf{W}_I$
- For a new user with features $\boldsymbol{a}_*$, we compute the latent factor $\boldsymbol{u}_* = \boldsymbol{a}_*\mathbf{W}_U$
- For a new item with features $\boldsymbol{b}_*$, we compute the latent factor $\boldsymbol{v}_* = \boldsymbol{b}_*\mathbf{W}_I$

Some Other Extensions of Matrix Factorization

Joint Matrix Factorization

- Can do joint matrix factorization of more than one matrices
- Consider two "ratings" matrices with the N users shared in both



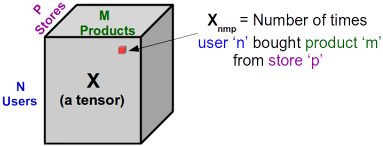
- Can assume the following matrix factorization

$$\mathbf{X}_1 \approx \mathbf{U}\mathbf{V}_1^\top \qquad \text{and} \qquad \mathbf{X}_2 \approx \mathbf{U}\mathbf{V}_2^\top$$

- Note that the user latent factor matrix $\mathbf{U}$ is shared in both factorizations
- Gives a way to learn features by combining multiple data sets (2 in this case)
- Can use the alternating optimization to solve for $\mathbf{U}$, $\mathbf{V}_1$ and $\mathbf{V}_2$

Tensor Factorization

- A "tensor" is a generalization of a matrix to more than two dimensions
- Consider a 3-dim (or 3-mode or 3-way) tensor $\mathbf{X}$ of size $N \times M \times P$



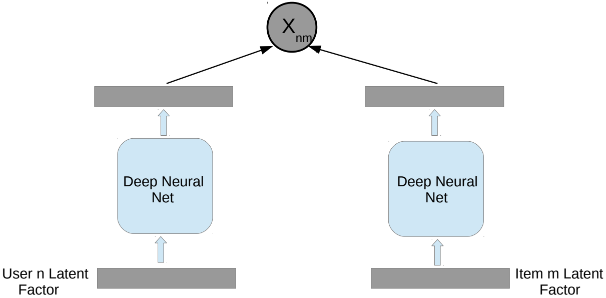
- We can model each entry of tensor $\mathbf{X}$ as

$$X_{nmp} \approx \boldsymbol{u}_n \odot \boldsymbol{v}_m \odot \boldsymbol{w}_p = \sum_{k=1}^{K} u_{nk} v_{mk} w_{pk}$$

- Can learn $\{\boldsymbol{u}_n\}_{n=1}^{N}, \{\boldsymbol{v}_m\}_{m=1}^{M}, \{\boldsymbol{w}_p\}_{p=1}^{P}$ using alternating optimization
- These K-dim. "embeddings" can be used as features for other tasks (e.g., tensor completion, computing similarities, etc.)
- The model also be easily extended to tensors having than 3 dimensions
- Several specialized algorithms for tensor factorization (CP/Tucker decomposition, etc.)

(And of course..) Deep Learning based Recommender Systems

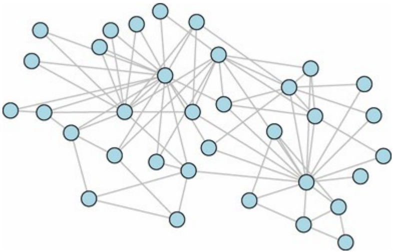
Basic idea: Matrix entries are nonlinear transformations of the latent factors $X_{nm} \approx f(\boldsymbol{u}_n)^\top f(\boldsymbol{v}_m)$

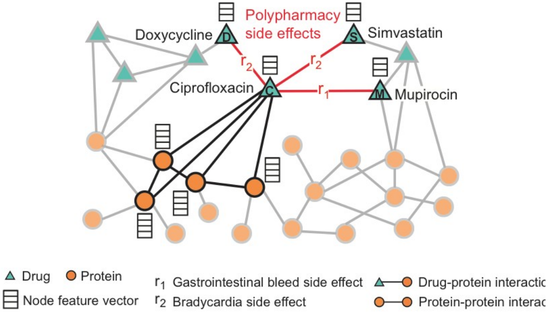


This above is a simple version. Many more sophisticated variants exist (posted a reference on the course webpage in case you are interested in deep learning methods for recommender systems)

Applications to Link Prediction in Graphs

- The user-item matrix is like a bipartite graph
- The matrix factorization ideas we saw today can also be used for any type of graph





- Thus we can get node embeddings as well as a way to do link prediction in such graphs

Right side picture courtesy: snap.standford.edu

Some Final Comments..

- Looked at some basic as well as some state-of-the-art approaches for recommendation systems
- Matrix factorization/completion is one of the dominant approaches (though not the only one) to solve the problem
- Other want to take into account other criteria such as freshness and diversity of recommendations
  - Don't want to keep recommending the similar items again and again
- Often helps to incorporate sources (e.g., meta data) other than just the user-item matrix
  - We saw some techniques already (e.g., inductive matrix completion)
- Temporal nature can also be incorporated (e.g., user and item latent factors may evolve in time)
- Still an ongoing area of active research