Matrix Factorization and Factorization Machines for Recommender Systems

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at Facebook, November 13, 2015

Introduction

- Matrix factorization (MF) and its extensions are now widely used in recommender systems
- In this talk I will briefly discuss three research works related to this topic

Outline

1. Parallel matrix factorization in shared-memory systems
2. Optimization algorithms for one-class matrix factorization
3. From matrix factorization to factorization machines and more
4. Conclusions

Outline

1. Parallel matrix factorization in shared-memory systems
2. Optimization algorithms for one-class matrix factorization
3. From matrix factorization to factorization machines and more
4. Conclusions

Matrix Factorization

- Matrix Factorization is an effective method for recommender systems (e.g., Netflix Prize and KDD Cup 2011)
- But training is slow.
- We developed a parallel MF package LIBMF for shared-memory systems

  `http://www.csie.ntu.edu.tw/~cjlin/libmf`
- Best paper award at ACM RecSys 2013

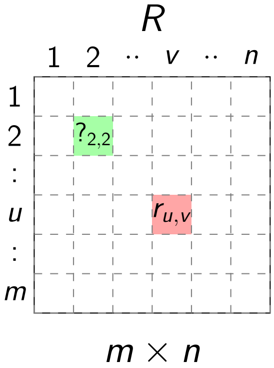
Matrix Factorization (Cont'd)

- A group of users give ratings to some items

User	Item	Rating
1	5	100
1	10	80
1	13	30
...	...	...
u	v	r
...	...	...

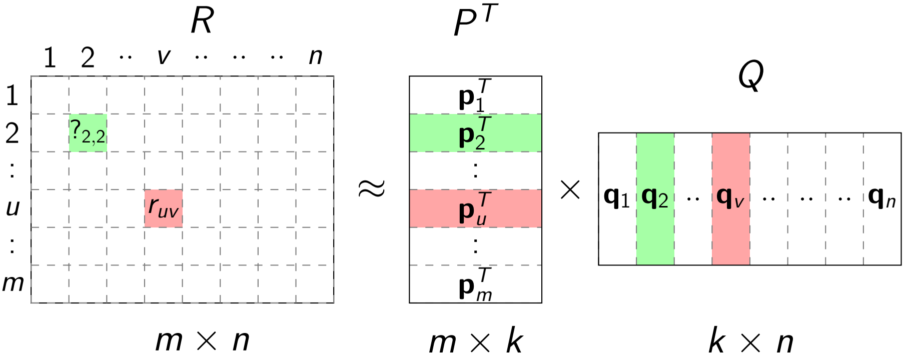
- The information can be represented by a rating matrix R

Matrix Factorization (Cont'd)



- m, n : numbers of users and items
- u, v : index for u_{th} user and v_{th} item
- $r_{u,v}$: u_{th} user gives a rating $r_{u,v}$ to v_{th} item

Matrix Factorization (Cont'd)



- k : number of latent dimensions
- $r_{u,v} = \mathbf{p}_u^T \mathbf{q}_v$
- $?_{2,2} = \mathbf{p}_2^T \mathbf{q}_2$

Matrix Factorization (Cont'd)

- A non-convex optimization problem:

$$\min_{P,Q} \sum_{(u,v)\in R} \left((r_{u,v} - \mathbf{p}_u^T \mathbf{q}_v)^2 + \lambda_P \|\mathbf{p}_u\|_F^2 + \lambda_Q \|\mathbf{q}_v\|_F^2 \right)$$

  λ_P and λ_Q are regularization parameters
- SG (Stochastic Gradient) is now a popular optimization method for MF
- It loops over ratings in the training set.

Matrix Factorization (Cont'd)

- SG update rule:

$$\mathbf{p}_u \leftarrow \mathbf{p}_u + \gamma \left(e_{u,v} \mathbf{q}_v - \lambda_P \mathbf{p}_u \right),$$
$$\mathbf{q}_v \leftarrow \mathbf{q}_v + \gamma \left(e_{u,v} \mathbf{p}_u - \lambda_Q \mathbf{q}_v \right)$$

  where

$$e_{u,v} \equiv r_{u,v} - \mathbf{p}_u^T \mathbf{q}_v$$

- SG is inherently sequential

SG for Parallel MF

After $r_{3,3}$ is selected, ratings in gray blocks cannot be updated

	1	2	3	4	5	6
1						
2						
3	$r_{3,1}$	$r_{3,2}$	$r_{3,3}$	$r_{3,4}$	$r_{3,5}$	$r_{3,6}$
4						
5						
6						$r_{6,6}$

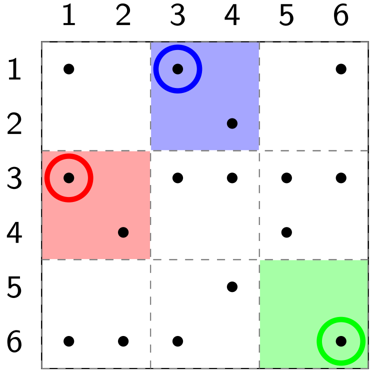
But $r_{6,6}$ can be used

- $r_{3,1} = \mathbf{p_3}^T \mathbf{q_1}$
- $r_{3,2} = \mathbf{p_3}^T \mathbf{q_2}$
- ..
- $r_{3,6} = \mathbf{p_3}^T \mathbf{q_6}$

- $r_{3,3} = \mathbf{p_3}^T \mathbf{q_3}$
  $r_{6,6} = \mathbf{p_6}^T \mathbf{q_6}$

SG for Parallel MF (Cont'd)

We can split the matrix to blocks and update those which don't share **p** or **q**



This concept is simple, but there are many issues to have a right implementation under the given architecture

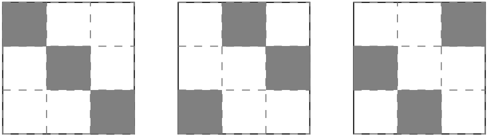
Our Approach in the Package LIBMF

- Parallelization (Zhuang et al., 2013; Chin et al., 2015a)
    - Effective block splitting to avoid synchronization time
    - Partial random method for the order of SG updates
- Adaptive learning rate for SG updates (Chin et al., 2015b)

  Details omitted due to time constraint

Block Splitting and Synchronization

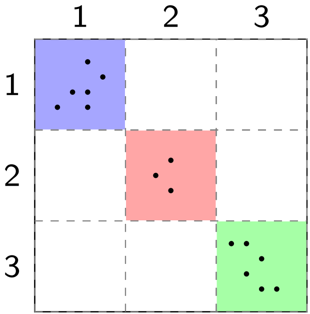
- A naive way for T nodes is to split the matrix to $T \times T$ blocks



- This is used in DSGD (Gemulla et al., 2011) for distributed systems, where communication cost is the main concern
- In distributed systems, it is difficult to move data or model

Block Splitting and Synchronization (Cont'd)

- For shared memory systems, synchronization becomes a concern



- Block 1: 20s
- Block 2: 10s
- Block 3: 20s

We have 3 threads

Thread	0→10	10→20
1	Busy	Busy
2	Busy	**Idle**
3	Busy	Busy

10s wasted!!

Lock-Free Scheduling

We split the matrix to enough blocks. For example, with two threads, we split the matrix to 4×4 blocks

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0 is the updated counter recording the number of updated times for each block

Lock-Free Scheduling (Cont'd)

Firstly, T_1 selects a block randomly

T_1 0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Lock-Free Scheduling (Cont'd)

For T_2, it selects a block neither green nor gray randomly

T_1 0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	T_2 0

Lock-Free Scheduling (Cont'd)

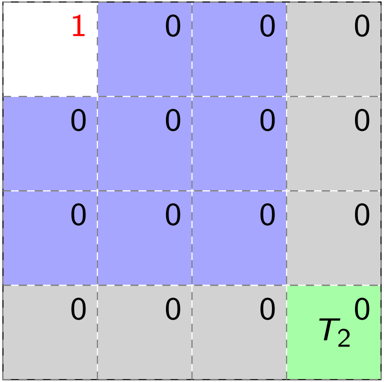
After T_1 finishes, the counter for the corresponding block is added by one

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0 T_2

Lock-Free Scheduling (Cont'd)

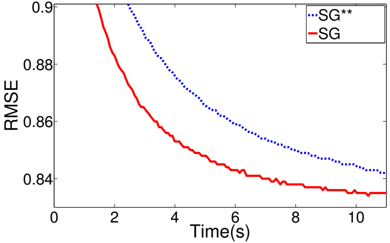
T_1 can select available blocks to update

Rule: select one that is least updated

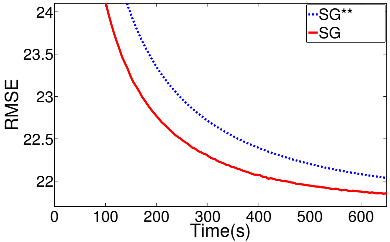


Lock-Free Scheduling (Cont'd)

SG: applying Lock-Free Scheduling
SG**: applying DSGD-like Scheduling



MovieLens 10M



Yahoo!Music

- MovieLens 10M: 18.71s $\rightarrow$ 9.72s (RMSE: 0.835)
- Yahoo!Music: 728.23s $\rightarrow$ 462.55s (RMSE: 21.985)

Memory Discontinuity

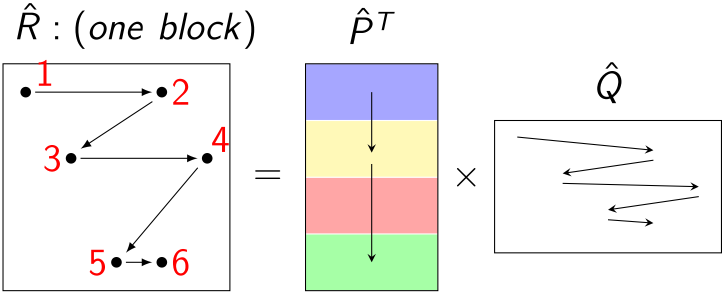
Discontinuous memory access can dramatically increase the training time. For SG, two possible update orders are

Update order	Advantages	Disadvantages
Random	Faster and stable	Memory discontinuity
Sequential	Memory continuity	Not stable

Our lock-free scheduling gives randomness, but the resulting code may not be cache friendly

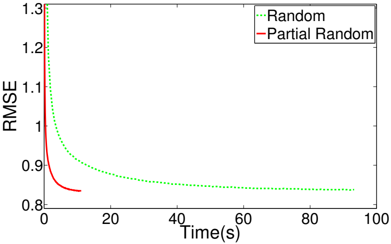
Partial Random Method

Our solution is that for each block, access both $\hat{R}$ and $\hat{P}$ continuously

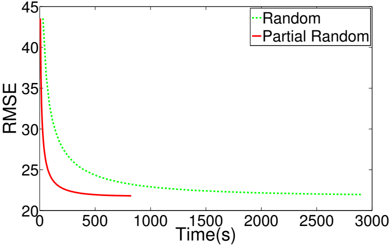


- Partial: sequential in **each block**
- Random: random when **selecting block**

Partial Random Method (Cont'd)



MovieLens 10M



Yahoo!Music

- The performance of Partial Random Method is better than that of Random Method

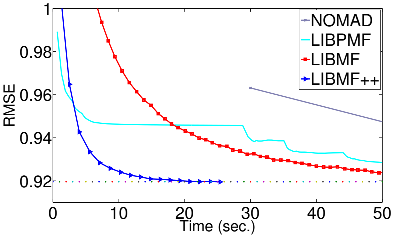
Experiments

State-of-the-art methods compared

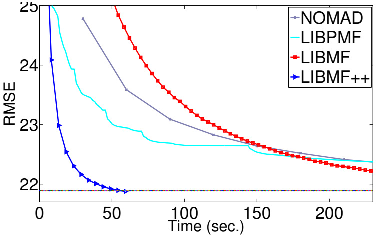
- `LIBPMF`: a parallel coordinate descent method (Yu et al., 2012)
- `NOMAD`: an asynchronous SG method (Yun et al., 2014)
- `LIBMF`: earlier version of LIBMF (Zhuang et al., 2013; Chin et al., 2015a)
- `LIBMF++`: with adaptive learning rates for SG (Chin et al., 2015c)

Details of data sets are omitted; the largest has 1.7B ratings.

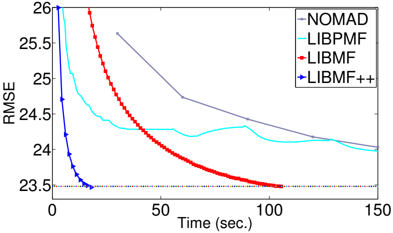
Results: $k = 100$



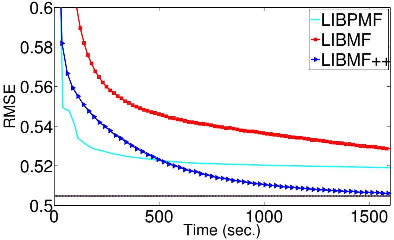
Netflix



Yahoo!Music



Webscope-R1



Hugewiki

Outline

1. Parallel matrix factorization in shared-memory systems
2. Optimization algorithms for one-class matrix factorization
3. From matrix factorization to factorization machines and more
4. Conclusions

One-class Matrix Factorization

- Some applications have only two possible ratings

  positive (1, watched) and

  negative (0, not-watched)
- One-class observation (i.e., implicit feedback)

  $\Rightarrow$ only part of positive actions are recorded

User	Item	Watched $\in \{0,1\}$
61	7	1
61	23	1
1647	128	1

- Past works include Pan et al. (2008); Hu et al. (2008); Pan and Scholz (2009); Li et al. (2010); Paquet and Koenigstein (2013)

Selection of Negative Samples

- One popular solution: treat some missing entries as negative

  Why? Most unknown entries are negative.

  $\Rightarrow$ a user cannot watch all the movies

$$\min_{P,Q} \sum_{(u,v)\in\Omega^+} C_{uv}(1-\mathbf{p}_u^T\mathbf{q}_v)^2 + \sum_{(u,v)\in\Omega^-} C_{uv}(0-\mathbf{p}_u^T\mathbf{q}_v)^2 + \lambda(\|P\|_F^2+\|Q\|_F^2)$$

- Ω^+: observed positive entries
- Ω^-: negative entries sampled from missing entries
- C_{uv}: weights

Two Ways to Select Negative Entries

We may use a subset or include all missing entries

- **Subsampled** :

$$|\Omega^-| = O(|\Omega^+|) \ll mn$$

  Full :

$$\Omega^- = \{(u, v) \mid (u, v) \notin \Omega^+\}$$

- **Subsampled** is just an approximation for **Full**
- **Full**: no need to worry about the selection
- **Full**: $O(mn)$ elements lead to a hard optimization problem

  Subsampled: existing MF techniques can be directly applied

Full for One-class Matrix Factorization

- Include all missing entries in Ω^-

$$\Omega = \Omega^+ \cup \Omega^- = \{1, \ldots, m\} \times \{1, \ldots, n\}$$

- Weighted matrix factorization:

$$\sum_{(i,j)\in\Omega^+} C_{uv}(A_{uv} - \mathbf{p}_u^T \mathbf{q}_v)^2 + \sum_{(i,j)\in\Omega^-} C_{uv}(0 - \mathbf{p}_u^T \mathbf{q}_v)^2 + \lambda(\|P\|_F^2 + \|Q\|_F^2)$$

- For most MF algorithms, the complexity is proportional to $O(|\Omega|)$. Now $|\Omega| = mn$ can be huge
- Therefore, even if **Full** gives better performance, it's not useful without efficient training techniques

New Optimization Techniques

Under certain conditions on C_{uv}, we reduce the mn term to Ω^+ in the optimization algorithms:

- ALS: Alternating Least Squares (ALS)
  The has been done by Pan and Scholz (2009)
- CD : Coordinate Descent
- SG : Stochastic Gradient

	ALS	CD	SG
rating-MF	$O(\lvert\Omega\rvert k^2+(m+n)k^3)$	$O(\lvert\Omega\rvert k)$	$O(\lvert\Omega\rvert k)$
Subsampled	$O(\lvert\Omega^+\rvert k^2+(m+n)k^3)$	$O(\lvert\Omega^+\rvert k)$	$O(\lvert\Omega^+\rvert k)$
Full-direct	$O(mnk^2+(m+n)k^3)$	$O(mnk)$	$O(mnk)$
Full-new	$O(\lvert\Omega^+\rvert k^2+(m+n)k^3)$	$O(\lvert\Omega^+\rvert k+(m+n)k^2)$	??

New Optimization Techniques (Cont'd)

Weights C_{uv} should satisfy certain conditions. Like Pan and Scholz (2009), we assume

$$C_{uv} = p_u q_v, \forall u, v \notin R.$$

This is often satisfied in practice. For example, we may have

$$C_{uv} \propto |\Omega_u^+|, \forall u, \text{ when } v \text{ is fixed}$$

where

$$|\Omega_u^+| = \# \text{ of user } u\text{'s observed entries}$$

New Optimization Techniques (Cont'd)

$$\sum_{(u,v)\in\Omega^+}(\cdots) + \sum_{(u,v)\in\Omega^-}(\cdots), u = 1, \ldots, m$$

can be written as

$$\sum\nolimits_{(u,v)\in\Omega^+}(\cdots + \text{ something from the 2nd summation})+$$

$$(\text{a term involving } u) \sum\nolimits_{v=1}^{n} (\text{a term involving } v), \forall u$$

The derivation is a bit complicated. Also some implementation issues must be carefully addressed.

Reducing the complexity of SG remains a challenging issue.

Comparison: **Subsampled** and **Full**

ml10m	nDCG @1	nDCG @10	nHLU	MAP	AUC
Subsampled	9.33	12.10	13.31	10.00	0.97293
Full	25.64	23.81	24.94	17.70	0.97372

netflix	nDCG @1	nDCG @10	nHLU	MAP	AUC
Subsampled	10.62	11.27	12.03	8.91	0.97224
Full	27.04	22.62	22.72	14.36	0.96879

For nDCG, nHLU and MAP, **Full** is much better than **Subsampled**. AUC isn't a good criterion as we now care more about top recommendations

Summary for One-class MF

- With the developed optimization techniques, the **Full** approach of treating all missing entries as negative becomes practical
- This work was done while I visited Microsoft

Outline

1. Parallel matrix factorization in shared-memory systems
2. Optimization algorithms for one-class matrix factorization
3. From matrix factorization to factorization machines and more
4. Conclusions

MF versus Classification/Regression

- MF solves

$$\min_{P,Q} \sum_{(u,v)\in R} \left(r_{u,v} - \mathbf{p}_u^T \mathbf{q}_v\right)^2$$

  Note that I omit the regularization term
- Ratings are the only given information
- This doesn't sound like a classification or regression problem
- In the last part of this talk we will make a connection and introduce FM (Factorization Machines)

Handling User/Item Features

- What if instead of user/item IDs we are given user and item features?
- Assume user u and item v have feature vectors

$$\mathbf{f}_u \in R^U \text{ and } \mathbf{g}_v \in R^V,$$

where

$$U \equiv \text{number of user features}$$
$$V \equiv \text{number of item features}$$

- How to use these features to build a model?

Handling User/Item Features (Cont'd)

- We can consider a regression problem where data instances are

$$\begin{array}{cc} \text{value} & \text{features} \\ \vdots & \vdots \\ r_{uv} & \begin{bmatrix} \mathbf{f}_u^T & \mathbf{g}_v^T \end{bmatrix} \\ \vdots & \vdots \end{array}$$

  and solve

$$\min_{\mathbf{w}} \sum_{u,v \in R} \left(R_{u,v} - \mathbf{w}^T \begin{bmatrix} \mathbf{f}_u \\ \mathbf{g}_v \end{bmatrix} \right)^2$$

Feature Combinations

- However, this does not take the interaction between users and items into account
- Following the concept of degree-2 polynomial mappings in SVM, we can generate new features

$$(f_u)_t(g_v)_s, t = 1, \dots, U, s = 1, \dots V$$

and solve

$$\min_{w_{t,s}, \forall t,s} \sum_{u,v \in R} (r_{u,v} - \sum_{t=1}^{U} \sum_{s=1}^{V} w_{t,s}(f_u)_t(g_v)_s)^2$$

Feature Combinations (Cont'd)

- This is equivalent to

$$\min_{W} \sum_{u,v \in R} (r_{u,v} - \mathbf{f}_u^T W \mathbf{g}_v)^2,$$

where

$$W \in R^{U \times V} \text{ is a matrix}$$

- If we have $\text{vec}(W)$ by concatenating W's columns, another form is

$$\min_{W} \sum_{u,v \in R} \left(r_{u,v} - \text{vec}(W)^T \begin{bmatrix} \vdots \\ (f_u)_t (g_v)_s \\ \vdots \end{bmatrix} \right)^2,$$

Feature Combinations (Cont'd)

- However, this setting fails for extremely sparse features
- Consider the most extreme situation. Assume we have

  user ID and item ID

  as features
- Then

$$U = m, J = n,$$
$$\mathbf{f}_i = [\underbrace{0, \ldots, 0}_{i-1}, 1, 0, \ldots, 0]^T$$

Feature Combinations (Cont'd)

- The optimal solution is

$$W_{u,v} = \begin{cases} r_{u,v}, & \text{if } u, v \in R \\ 0, & \text{if } u, v \notin R \end{cases}$$

- We can never predict

$$r_{u,v}, u, v \notin R$$

Factorization Machines

- The reason why we cannot predict unseen data is because in the optimization problem

$$\#\ \text{variables} = mn \gg \#\ \text{instances}\ = |R|$$

- Overfitting occurs
- Remedy: we can let

$$W \approx P^T Q,$$

where P and Q are low-rank matrices. This becomes matrix factorization

Factorization Machines (Cont'd)

- This can be generalized to sparse user and item features

$$\min_{u,v \in R} (R_{u,v} - \mathbf{f}_u^T P^T Q \mathbf{g}_v)^2$$

- That is, we think

$$P\mathbf{f}_u \text{ and } Q\mathbf{g}_v$$

  are latent representations of user u and item v, respectively
- This becomes factorization machines (Rendle, 2010)

Factorization Machines (Cont'd)

- Similar ideas have been used in other places such as Stern et al. (2009)
- We see that such ideas can be used for not only recommender systems.
- They may be useful for any classification problems with very sparse features

Field-aware Factorization Machines

- We have seen that FM is useful to handle highly sparse features such as user IDs
- What if we have more than two ID fields?
- For example, in CTR prediction for computational advertising, we may have

value	features
⋮	⋮
CTR	user ID, Ad ID, site ID
⋮	⋮

Field-aware Factorization Machines (Cont'd)

- FM can be generalized to handle different interactions between fields

  Two latent matrices for user ID and Ad ID
  Two latent matrices for user ID and site ID
  ⋮

- This becomes FFM: field-aware factorization machines (Rendle and Schmidt-Thieme, 2010)

FFM for CTR Prediction

- It's used by Jahrer et al. (2012) to win the 2nd prize of KDD Cup 2012
- Recently my students used FFM to win two Kaggle competitions
- After we used FFM to win the first competition, in the second competition all top teams use FFM
- Note that for CTR prediction, logistic rather than squared loss is used

Discussion

- How to decide which field interactions to use?
- If features are not extremely sparse, can the result still be better than degree-2 polynomial mappings?
  Note that we lose the convexity here
- We have a software LIBFFM for public use
  `http://www.csie.ntu.edu.tw/~cjlin/libffm`

Outline

1. Parallel matrix factorization in shared-memory systems
2. Optimization algorithms for one-class matrix factorization
3. From matrix factorization to factorization machines and more
4. Conclusions

Discussion and Conclusions

- From my limited experience on recommender systems, I feel that the practical use is very problem dependent
- For example, sometimes many features are available, but sometimes you only have ratings
- Developing general algorithms becomes difficult. An algorithm may be useful only for certain scenarios

Discussion and Conclusions (Cont'd)

- This situation is different from data classification, where the process is more standardized
- I am still learning different aspects of recommender systems. Your comments/suggestions are very welcome

Acknowledgments

Collaborators for works mentioned in this talk:

- National Taiwan University
    - Wei-Sheng Chin
    - Yu-Chin Juan
    - Bo-Wen Yuan
    - Yong Zhuang
- UT Austin
    - Hsiang-Fu Yu
- Microsoft
    - Misha Bilenko