Matrix Factorization and Factorization Machines for Recommender Systems

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at SDM workshop on Machine Learning Methods on Recommender Systems, May 2, 2015

Outline

1. Matrix factorization
2. Factorization machines
3. Conclusions

In this talk I will briefly discuss two related topics

- Fast matrix factorization (MF) in shared-memory systems
- Factorization machines (FM) for recommender systems and classification/regression

Note that MF is a special case of FM

Outline

1. Matrix factorization
   - Introduction and issues for parallelization
   - Our approach in the package LIBMF
2. Factorization machines
3. Conclusions

Outline

1. Matrix factorization
   - Introduction and issues for parallelization
   - Our approach in the package LIBMF
2. Factorization machines
3. 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

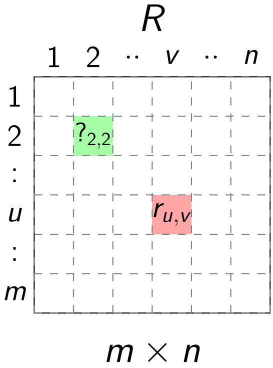
Matrix Factorization (Cont'd)

- For recommender systems: a group of users give ratings to some items

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

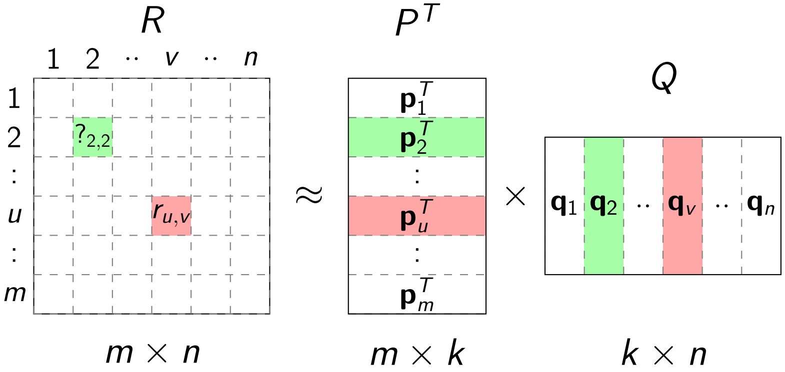
- The information can be represented by a rating matrix R

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:

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

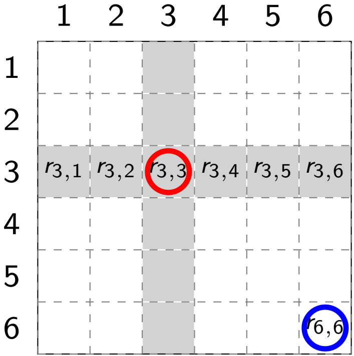
  where

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

- SG is inherently sequential

SG for Parallel MF

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



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

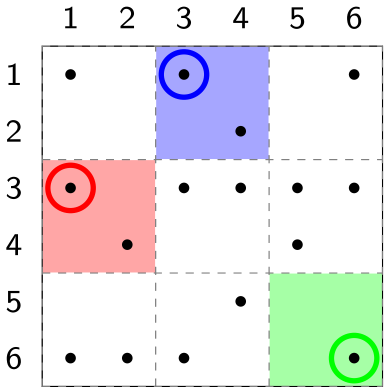
- $r_{3,3} = \mathbf{p_3}^T \mathbf{q_3}$
  $r_{6,6} = \mathbf{p_6}^T \mathbf{q_6}$

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

SG for Parallel MF (Cont'd)

We can split the matrix to blocks.

Then use threads to update the blocks where ratings in different blocks don't share **p** or **q**



SG for Parallel MF (Cont'd)

- This concept of splitting data to independent blocks seems to work
- However, there are many issues to have a right implementation under the given architecture

Outline

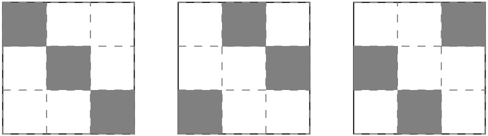
Our approach in the package LIBMF

- 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

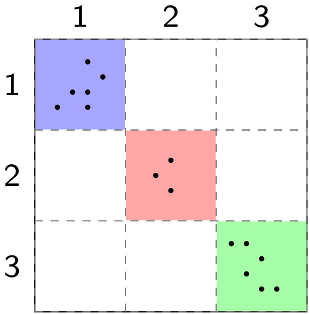
- A naive way for T nodes is to split the matrix to $T \times T$ blocks



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

Block Splitting and Synchronization (Cont'd)

- However, for shared memory systems, synchronization is 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

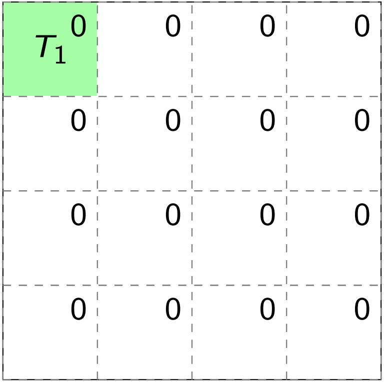
We split the matrix to enough blocks. For example, with two threads, we split the matrix to 4×4 blocks

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

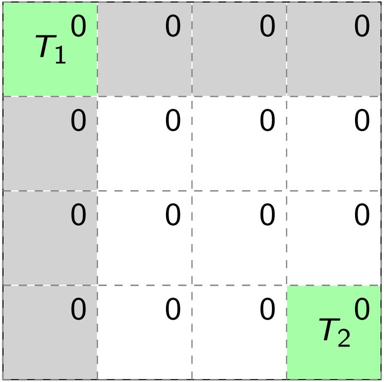
Lock-Free Scheduling (Cont'd)

Firstly, T_1 selects a block randomly



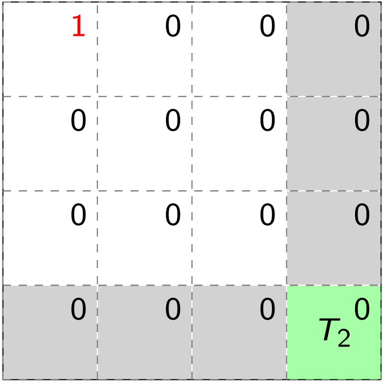
Lock-Free Scheduling (Cont'd)

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



Lock-Free Scheduling (Cont'd)

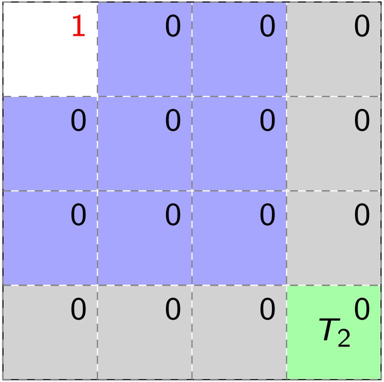
After T_1 finishes, the counter for the corresponding block is added by one



Lock-Free Scheduling (Cont'd)

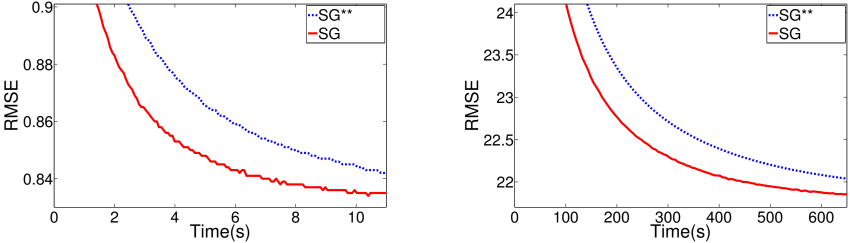
T_1 can select available blocks to update

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

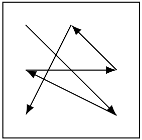
- MovieLens 10M: 18.71s → 9.72s (RMSE: 0.835)
- Yahoo!Music: 728.23s → 462.55s (RMSE: 21.985)

Memory Discontinuity

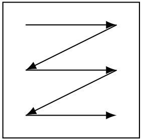
Discontinuous memory access can dramatically increase the training time. For SG, two possible update orders are

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

Random

R 

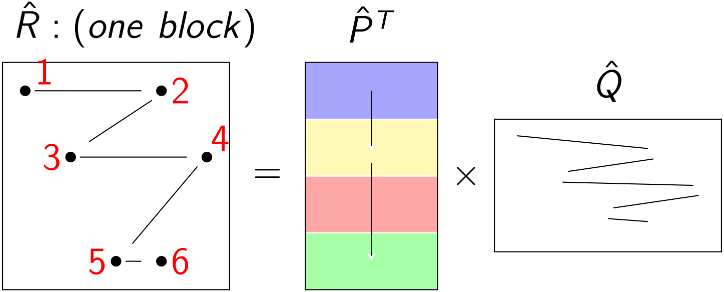
Sequential

R 

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

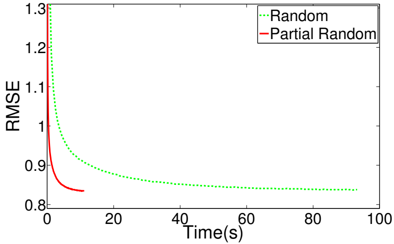
Partial Random Method

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

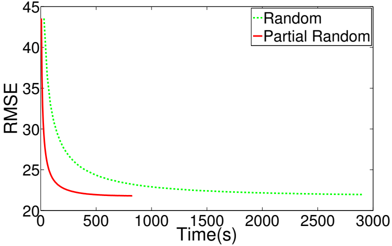


- Partial: sequential in **each block**
- Random: random when **selecting block**

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

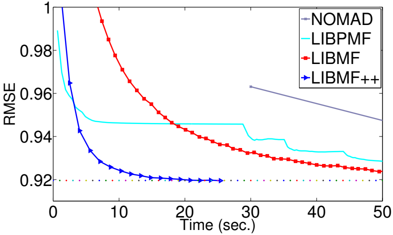
- `LIBPMF`: a parallel coordinate descent method (Yu et al., 2012)
- `NOMAD`: an asynchronous SG method (Yun et al., 2014)
- `LIBMF`: earlier version of LIBMF (Zhuang et al., 2013; Chin et al., 2015a)
- `LIBMF++`: with adaptive learning rates for SG (Chin et al., 2015c)

Experiments (Cont'd)

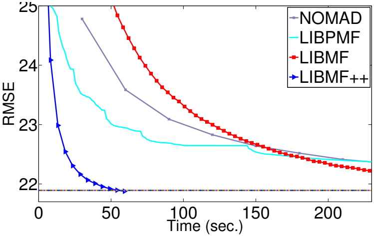
Data Set	m	n	#ratings
`Netflix`	2,649,429	17,770	99,072,112
`Yahoo!Music`	1,000,990	624,961	252,800,275
`Webscope-R1`	1,948,883	1,101,750	104,215,016
`Hugewiki`	39,706	25,000,000	1,703,429,136

- Due to machine capacity, `Hugewiki` here is about half of the original
- $k = 100$

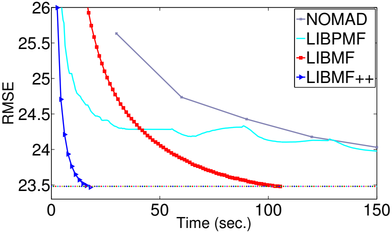
Experiments (Cont'd)



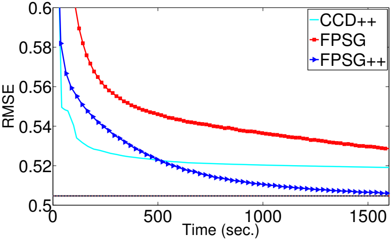
Netflix



Yahoo!Music



Webscope-R1



Hugewiki

Non-negative Matrix Factorization (NMF)

- Our method has been extended to solve NMF

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

subject to $P_{i,u} \geq 0, Q_{i,v} \geq 0, \forall i, u, v$

Outline

MF and 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 second 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

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

- 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
- Note that we are approximating the rating $r_{u,v}$ of user u and item v
- Let

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

- Then

$$\mathbf{f}_u \in R^U, u = 1, \ldots, m,$$
$$\mathbf{g}_v \in R^V, v = 1, \ldots, n$$

Feature Combinations (Cont'd)

- Following the concept of degree-2 polynomial mappings in SVM, we can generate new features

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

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, Herbrich, and Graepel (2009)
- In summary, we connect MF and classification/regression by the following settings
  - We need combination of different feature types (e.g., user, item, etc)
  - However, overfitting occurs if features are very sparse
  - We use product of low-rank matrices to avoid overfitting

Factorization Machines (Cont'd)

- 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, 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`

Experiments

- We see that
$$W \Rightarrow P^T Q$$
reduces the number of variables
- What if we map
$$\begin{bmatrix} \vdots \\ (f_u)_t (g_v)_s \\ \vdots \end{bmatrix} \Rightarrow \text{ a shorter vector}$$
to reduce the number of features/variables

Experiments (Cont'd)

- However, we may have something like

$$
\begin{aligned}
(r_{1,2} - W_{1,2})^2 &\Rightarrow (r_{1,2} - \bar{w}_1)^2 && (1)\\
(r_{1,4} - W_{1,4})^2 &\Rightarrow (r_{1,4} - \bar{w}_2)^2 \\
(r_{2,1} - W_{2,1})^2 &\Rightarrow (r_{2,1} - \bar{w}_3)^2 \\
(r_{2,3} - W_{2,3})^2 &\Rightarrow (r_{2,3} - \bar{w}_1)^2 && (2)
\end{aligned}
$$

- Clearly, there is no reason why (1) and (2) should share the same variable $\bar{w}_1$
- In contrast, in MF, we connect $r_{1,2}$ and $r_{1,3}$ through $\mathbf{p}_1$

Experiments (Cont'd)

- A simple comparison on MovieLens
  # training: 9,301,274, # test: 698,780, # users: 71,567, # items: 65,133
- Results of MF: RMSE = 0.836
- Results of Poly-2 + Hashing:
  RMSE = 1.14568 (10^6 bins), 3.62299 (10^8 bins), 3.76699 (all pairs)
- We can clearly see that MF is much better

Outline

Conclusions

- In this talk we have talked about MF and FFM
- MF is a mature technique, so we investigate its fast training
- FFM is relatively new. We introduce its basic concepts and practical use

Acknowledgments

The following students have contributed to works mentioned in this talk

- Wei-Sheng Chin
- Yu-Chin Juan
- Bo-Wen Yuan
- Yong Zhuang