Matrix Factorization and Factorization Machines for Recommender Systems

Chih-Jen Lin
Department of Computer Science
National Taiwan University

Talk at Facebook, November 13, 2015
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
Parallel matrix factorization in shared-memory systems

Optimization algorithms for one-class matrix factorization

From matrix factorization to factorization machines and more

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

<table>
<thead>
<tr>
<th>User</th>
<th>Item</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>13</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The information can be represented by a rating matrix $R$
Matrix Factorization (Cont’d)

\[ R \]

\[
\begin{array}{cccccc}
1 & 2 & \cdots & v & \cdots & n \\
1 & \text{??} & \text{r}_{u,v} \\
2 & \text{?} & \text{u} & \text{v} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
m & \text{m} & \times & \text{n} & \text{m} & \times & \text{n} \\
\end{array}
\]

- \( m, n \): 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)

\[ R \approx P^T \times Q \]

- \( k \): number of latent dimensions
- \( r_{uv} = p_u^T q_v \)
- \( ?_{2,2} = p_2^T q_2 \)
Matrix Factorization (Cont’d)

- A non-convex optimization problem:

\[
\min_{P,Q} \sum_{(u,v) \in R} \left( (r_{u,v} - p_u^T q_v)^2 + \lambda_P \| p_u \|_F^2 + \lambda_Q \| q_v \|_F^2 \right)
\]

- \( \lambda_P \) and \( \lambda_Q \) 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:

\[ p_u \leftarrow p_u + \gamma (e_{u,v} q_v - \lambda_P p_u), \]
\[ q_v \leftarrow q_v + \gamma (e_{u,v} p_u - \lambda_Q q_v) \]

where

\[ e_{u,v} \equiv r_{u,v} - p_u^T q_v \]

- SG is inherently sequential
SG for Parallel MF

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$r_{3,1}$</td>
<td>$r_{3,2}$</td>
<td>$r_{3,3}$</td>
<td>$r_{3,4}$</td>
<td>$r_{3,5}$</td>
<td>$r_{3,6}$</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

- $r_{3,1} = p_3^T q_1$
- $r_{3,2} = p_3^T q_2$
- $\ldots$
- $r_{3,6} = p_3^T q_6$
- $r_{6,6} = p_6^T q_6$
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.

![Diagram showing block splitting]

- 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.
For shared memory systems, synchronization becomes a concern.

Block Splitting and Synchronization (Cont’d)

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

We have 3 threads

<table>
<thead>
<tr>
<th>Thread</th>
<th>0→10</th>
<th>10→20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Busy</td>
<td>Busy</td>
</tr>
<tr>
<td>2</td>
<td>Busy</td>
<td>Idle</td>
</tr>
<tr>
<td>3</td>
<td>Busy</td>
<td>Busy</td>
</tr>
</tbody>
</table>

10s wasted!!
Lock-Free Scheduling

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

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**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Lock-Free Scheduling (Cont’d)

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

\[
\begin{array}{cccc}
T_1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & T_2 \end{array}
\]
Lock-Free Scheduling (Cont’d)

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

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & T_2^0 \\
\end{array}
\]
$T_1$ can select available blocks to update

**Rule:** select one that is least updated

$$
\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & T_2
\end{array}
$$
Parallel matrix factorization in shared-memory systems

Lock-Free Scheduling (Cont’d)

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

![Graphs](image)

**MovieLens 10M**
- MovieLens 10M: 18.71s → **9.72s** (RMSE: 0.835)

**Yahoo!Music**
- Yahoo!Music: 728.23s → **462.55s** (RMSE: 21.985)

Chih-Jen Lin (National Taiwan Univ.)
Discontinuous memory access can dramatically increase the training time. For SG, two possible update orders are:

<table>
<thead>
<tr>
<th>Update order</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>Faster and stable</td>
<td>Memory discontinuity</td>
</tr>
<tr>
<td>Sequential</td>
<td>Memory continuity</td>
<td>Not stable</td>
</tr>
</tbody>
</table>

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.

$\hat{R} : (\text{one block})$

Partial: sequential in each block
Random: random when selecting block
Partial Random Method (Cont’d)

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

MovieLens 10M

Yahoo!Music
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)
  - ⇒ only **part of positive actions** are recorded

<table>
<thead>
<tr>
<th>User</th>
<th>Item</th>
<th>Watched ∈ {0, 1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>61</td>
<td>23</td>
<td>1</td>
</tr>
<tr>
<td>1647</td>
<td>128</td>
<td>1</td>
</tr>
</tbody>
</table>

- 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.
⇒ a user cannot watch all the movies

\[
\min_{P,Q} \sum_{(u,v) \in \Omega^+} C_{uv} (1 - p_u^T q_v)^2 + \sum_{(u,v) \in \Omega^-} C_{uv} (0 - p_u^T q_v)^2
\]

\[+ \lambda (\|P\|_F^2 + \|Q\|_F^2)\]

- \(\Omega^+\): observed positive entries
- \(\Omega^-\): 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**:

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

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

- **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 \( \Omega^- \)

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

- Weighted matrix factorization:

\[
\sum_{(i,j) \in \Omega^+} C_{uv}(A_{uv} - p_u^T q_v)^2 + \sum_{(i,j) \in \Omega^-} C_{uv}(0 - p_u^T q_v)^2 + \lambda (\|P\|_F^2 + \|Q\|_F^2)
\]

- 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 $\Omega^+$ in the optimization algorithms:

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

<table>
<thead>
<tr>
<th>Optimization Algorithm</th>
<th>ALS</th>
<th>CD</th>
<th>SG</th>
</tr>
</thead>
<tbody>
<tr>
<td>rating-MF</td>
<td>$O(</td>
<td>\Omega</td>
<td>k^2 + (m+n)k^3)$</td>
</tr>
<tr>
<td>Subsampled</td>
<td>$O(</td>
<td>\Omega^+</td>
<td>k^2 + (m+n)k^3)$</td>
</tr>
<tr>
<td>Full-direct</td>
<td>$O(mnk^2 + (m+n)k^3)$</td>
<td>$O(mnk)$</td>
<td>$O(mnk)$</td>
</tr>
<tr>
<td>Full-new</td>
<td>$O(</td>
<td>\Omega^+</td>
<td>k^2 + (m+n)k^3)$</td>
</tr>
</tbody>
</table>
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

\[
\sum_{(u,v) \in \Omega^+} (\cdots + \text{something from the 2nd summation}) + (\text{a term involving } u) \sum_{v=1}^{n} (\text{a term involving } v), \ \forall u
\]

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

<table>
<thead>
<tr>
<th></th>
<th>ml10m nDCG</th>
<th>nHLU</th>
<th>MAP</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>@1 @10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subsampled</td>
<td>9.33 12.10</td>
<td>13.31</td>
<td>10.00</td>
<td>0.97293</td>
</tr>
<tr>
<td>Full</td>
<td>25.64 23.81</td>
<td>24.94</td>
<td>17.70</td>
<td>0.97372</td>
</tr>
<tr>
<td></td>
<td>netflix nDCG</td>
<td>nHLU</td>
<td>MAP</td>
<td>AUC</td>
</tr>
<tr>
<td></td>
<td>@1 @10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subsampled</td>
<td>10.62 11.27</td>
<td>12.03</td>
<td>8.91</td>
<td>0.97224</td>
</tr>
<tr>
<td>Full</td>
<td>27.04 22.62</td>
<td>22.72</td>
<td>14.36</td>
<td>0.96879</td>
</tr>
</tbody>
</table>

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.
Optimization algorithms for one-class matrix factorization

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

\[
\min_{P,Q} \sum_{(u,v) \in R} \left( r_{u,v} - p_u^T q_v \right)^2
\]

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 $f_u \in \mathbb{R}^U$ and $g_v \in \mathbb{R}^V$, where
  - $U \equiv$ number of user features
  - $V \equiv$ 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

\[ \begin{align*}
\text{value features} & \\
\vdots & \\
r_{uv} & = \begin{bmatrix} f_u^T \\ g_v^T \end{bmatrix}
\end{align*} \]

and solve

\[
\min_w \sum_{u,v \in R} \left( R_{u,v} - w^T \begin{bmatrix} f_u \\ g_v \end{bmatrix} \right)^2
\]
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

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

and solve

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

$$\min_W \sum_{u,v \in R} (r_{u,v} - f_u^T W g_v)^2,$$

where

$$W \in R^{U \times V}$$

is a matrix.

If we have 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

  \[ U = m, \quad J = n, \]
  \[ f_i = [0, \ldots, 0, 1, 0, \ldots, 0]^T \]
  \[ \underbrace{0, \ldots, 0}_{i-1}, 1, 0, \ldots, 0 \]
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

\[r_{u,v}, u, v \notin R\]
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
This can be generalized to sparse user and item features

\[
\min_{u,v \in R} (R_{u,v} - f_u^T P^T Q g_v)^2
\]

That is, we think

\[ Pf_u \text{ and } Q g_v \]

are latent representations of user \( u \) and item \( v \), respectively

This becomes factorization machines (Rendle, 2010)
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)
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.
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.
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