Lecture 15: Matrix Factorization

Hsuan-Tien Lin (林軒田)
htlin@csie.ntu.edu.tw

Department of Computer Science & Information Engineering
National Taiwan University (國立台灣大學資訊工程系)
Matrix Factorization

Roadmap

1. Embedding Numerous Features: Kernel Models
2. Combining Predictive Features: Aggregation Models
3. Distilling Implicit Features: Extraction Models

Lecture 14: Radial Basis Function Network

- Linear aggregation of distance-based similarities using \( k \)-Means clustering for prototype finding

Lecture 15: Matrix Factorization

- Linear Network Hypothesis
- Basic Matrix Factorization
- Stochastic Gradient Descent
- Summary of Extraction Models
Recommender System Revisited

- **data**: how ‘many users’ have rated ‘some movies’
- **skill**: predict how a user would rate an unrated movie
Recommender System Revisited

- **data**: how ‘many users’ have rated ‘some movies’
- **skill**: predict how a user would rate an unrated movie

### A Hot Problem

- competition held by Netflix in 2006
  - 100,480,507 ratings that 480,189 users gave to 17,770 movies
  - 10% improvement = 1 million dollar prize
Recommender System Revisited

- **data**: how ‘many users’ have rated ‘some movies’
- **skill**: predict how a user would rate an unrated movie

A Hot Problem

- competition held by Netflix in 2006
  - 100,480,507 ratings that 480,189 users gave to 17,770 movies
  - 10% improvement = 1 million dollar prize
- data $D_m$ for $m$-th movie:
  $$\{(\tilde{x}_n = (n), y_n = r_{nm}): \text{user } n \text{ rated movie } m\}$$
Recommender System Revisited

**data** → **ML** → **skill**

- **data**: how ‘many users’ have rated ‘some movies’
- **skill**: predict how a user would rate an unrated movie

**A Hot Problem**

- competition held by Netflix in 2006
  - 100,480,507 ratings that 480,189 users gave to 17,770 movies
  - 10% improvement = 1 million dollar prize
- data $D_m$ for $m$-th movie:
  \[ \{(\tilde{x}_n = (n), y_n = r_{nm}) : \text{user } n \text{ rated movie } m\} \]
  —abstract feature $\tilde{x}_n = (n)$
Recommender System Revisited

- **Data**: how ‘many users’ have rated ‘some movies’
- **Skill**: predict how a user would rate an unrated movie

A Hot Problem

- Competition held by Netflix in 2006
  - 100,480,507 ratings that 480,189 users gave to 17,770 movies
  - 10% improvement = 1 million dollar prize
- Data $D_m$ for $m$-th movie:
  \[ \{(\tilde{x}_n = (n), y_n = r_{nm}) : \text{user } n \text{ rated movie } m\} \]
  —Abstract feature $\tilde{x}_n = (n)$

How to learn our preferences from data?
Binary Vector Encoding of Categorical Feature

\[ \tilde{x}_n = (n) : \text{user IDs, such as 1126, 5566, 6211, \ldots} \]
Binary Vector Encoding of Categorical Feature

\[ \tilde{x}_n = (n) : \text{user IDs, such as } 1126, 5566, 6211, \ldots \]
—called \textit{categorical} features
Matrix Factorization

Linear Network Hypothesis

Binary Vector Encoding of Categorical Feature

$$\tilde{x}_n = (n):$$ user IDs, such as 1126, 5566, 6211, . . .—called **categorical** features

- **categorical** features, e.g.
  - IDs
  - blood type: A, B, AB, O
Binary Vector Encoding of Categorical Feature

\[ \tilde{x}_n = (n) \]: user IDs, such as 1126, 5566, 6211, ... —called **categorical** features

- **categorical** features, e.g.
  - IDs
  - blood type: A, B, AB, O
  - programming languages: C, C++, Java, Python, ...
Binary Vector Encoding of Categorical Feature

\[ \tilde{x}_n = (n) : \text{user IDs, such as 1126, 5566, 6211, \ldots} \]
—called **categorical** features

- **categorical** features, e.g.
  - IDs
  - blood type: A, B, AB, O
  - programming languages: C, C++, Java, Python, \ldots
- many ML models operate on **numerical** features
\[ \tilde{x}_n = (n) : \text{user IDs, such as 1126, 5566, 6211, \ldots} \]
—called **categorical** features

- **categorical** features, e.g.
  - IDs
  - blood type: A, B, AB, O
  - programming languages: C, C++, Java, Python, \ldots
- many ML models operate on **numerical** features
  - linear models
Binary Vector Encoding of Categorical Feature

\[ \tilde{x}_n = (n) \]: user IDs, such as 1126, 5566, 6211, ... —called \textit{categorical} features

- \textbf{categorical} features, e.g.
  - IDs
  - blood type: A, B, AB, O
  - programming languages: C, C++, Java, Python, ...
- many ML models operate on \textit{numerical} features
  - \textit{linear} models
  - \textit{extended linear} models such as NNet
Binary Vector Encoding of Categorical Feature

\[ \tilde{x}_n = (n) \]: user IDs, such as 1126, 5566, 6211, ... —called \textit{categorical} features

- \textit{categorical} features, e.g.
  - IDs
  - blood type: A, B, AB, O
  - programming languages: C, C++, Java, Python, ...

- many ML models operate on \textit{numerical} features
  - linear models
  - extended linear models such as NNet

—except for \textit{decision trees}
Binary Vector Encoding of Categorical Feature

\[ \tilde{x}_n = (n) : \text{user IDs, such as 1126, 5566, 6211, \ldots} \]
—called \textit{categorical} features

- \textbf{categorical} features, e.g.
  - IDs
  - \textit{blood type}: A, B, AB, O
  - \textit{programming languages}: C, C++, Java, Python, \ldots
- many ML models operate on \textit{numerical} features
  - \textit{linear} models
  - \textit{extended linear} models such as NNet
—except for \textit{decision trees}
- need: \textit{encoding (transform)} from \textit{categorical} to \textit{numerical}
Binary Vector Encoding of Categorical Feature

\[ \tilde{x}_n = (n) \]: user IDs, such as 1126, 5566, 6211, \ldots
—called \textit{categorical} features

- \textit{categorical} features, e.g.
  - IDs
  - blood type: A, B, AB, O
  - programming languages: C, C++, Java, Python, \ldots
- many ML models operate on \textit{numerical} features
  - \textit{linear} models
  - \textit{extended linear} models such as NNet
—except for \textit{decision trees}
- need: \textit{encoding (transform)} from \textit{categorical} to \textit{numerical}

\textbf{binary vector encoding:}

\[
A = [1 \ 0 \ 0 \ 0]^T, \quad B = [0 \ 1 \ 0 \ 0]^T,
AB = [0 \ 0 \ 1 \ 0]^T, \quad O = [0 \ 0 \ 0 \ 1]^T
\]
Feature Extraction from Encoded Vector

Encoded data $D_m$ for $m$-th movie:

$$\{(x_n = \text{BinaryVectorEncoding}(n), y_n = r_{nm}): \text{user } n \text{ rated movie } m\}$$
Feature Extraction from Encoded Vector

encoded data $\mathcal{D}_m$ for $m$-th movie:

$$\left\{ (x_n = \text{BinaryVectorEncoding}(n), y_n = r_{nm}) : \text{user } n \text{ rated movie } m \right\}$$

or, joint data $\mathcal{D}$

$$\left\{ (x_n = \text{BinaryVectorEncoding}(n), y_n = [r_{n1} \ ? \ ? \ r_{n4} \ r_{n5} \ \ldots \ r_{nM}]^T) \right\}$$
encoded data $D_m$ for $m$-th movie:

$$\left\{ (x_n = \text{BinaryVectorEncoding}(n), y_n = r_{nm}) : \text{user } n \text{ rated movie } m \right\}$$

or, joint data $D$

$$\left\{ (x_n = \text{BinaryVectorEncoding}(n), y_n = [r_{n1} \ ? \ ? \ r_{n4} \ r_{n5} \ \ldots \ r_{nM}]^T) \right\}$$

idea: try **feature extraction** using $N\tilde{n}-M$ NNet without all $x_0^{(\ell)}$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

$$x_1 \xrightarrow{W_{ni}^{(1)}} \tanh \xrightarrow{W_{im}^{(2)}} y_1 \approx y_1$$

$$x_2 \xrightarrow{W_{ni}^{(1)}} \tanh \xrightarrow{W_{im}^{(2)}} y_2 \approx y_2$$

$$x_3 \xrightarrow{W_{ni}^{(1)}} \tanh \xrightarrow{W_{im}^{(2)}} y_3 \approx y_3$$

$$x_4 \xrightarrow{W_{ni}^{(1)}} \tanh \xrightarrow{W_{im}^{(2)}} y_4 \approx y_4$$

$$\approx y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$
**Feature Extraction from Encoded Vector**

encoded data $D_m$ for $m$-th movie:

$$\{(x_n = \text{BinaryVectorEncoding}(n), y_n = r_{nm}): \text{user } n \text{ rated movie } m\}$$

or, joint data $D$

$$\{(x_n = \text{BinaryVectorEncoding}(n), y_n = [r_{n1} \ ? \ ? \ r_{n4} \ r_{n5} \ \ldots \ \ r_{nM}]^T)\}$$

idea: try feature extraction using $N-\tilde{d}-M$ NNet without all $x_0^{(\ell)}$

$$x = \begin{array}{cccc}
    x_1 & W_{ni}^{(1)} & \text{tanh} & \approx y_1 \\
    x_2 & \text{tanh} & W_{im}^{(2)} & \approx y_2 \\
    x_3 & \text{tanh} & \text{y}_3 \\
    x_4 & & & \approx y_3
\end{array}$$

is tanh necessary? :-)}
Matrix Factorization

'Linear Network' Hypothesis

\[
\begin{align*}
\mathbf{x} & = \begin{bmatrix}
    x_1 \\
    x_2 \\
    x_3 \\
    x_4 \\
\end{bmatrix} \\
\mathbf{y} & = \begin{bmatrix}
    y_1 \\
    y_2 \\
    y_3 \\
\end{bmatrix}
\end{align*}
\]

\[
\mathbf{x} = \text{BinaryVectorEncoding}(n), \mathbf{y} = [r_{n1} \ ? \ ? \ r_{n4} \ r_{n5} \ \ldots \ r_{nM}]^T
\]

Hsuan-Tien Lin (NTU CSIE)
Matrix Factorization

**Linear Network Hypothesis**

\[ \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} w_{ni}^{(1)} \\ w_{im}^{(2)} \end{pmatrix} \approx \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \mathbf{y} \]

\[
\left\{ (\mathbf{x}_n = \text{BinaryVectorEncoding}(n), \mathbf{y}_n = \begin{bmatrix} r_{n1} \\ \vdots \\ r_{n4} \\ r_{n5} \\ \vdots \\ r_{nM} \end{bmatrix}^T) \right\}
\]

- rename: for \( w_{ni}^{(1)} \) and for \( w_{im}^{(2)} \)
‘Linear Network’ Hypothesis

\[ \mathbf{X} = \mathbf{V}^T : \mathbf{w}_{ni}^{(1)} \approx y_1 \approx y_2 \approx y_3 = \mathbf{y} \]

\[ \{(\mathbf{x}_n = \text{BinaryVectorEncoding}(n), \mathbf{y}_n = [r_{n1} \ ？ \ ？ \ r_{n4} \ r_{n5} \ \ldots \ r_{nM}]^T)\} \]

- rename: \( V^T \) for \( \mathbf{w}_{ni}^{(1)} \) and \( W \) for \( \mathbf{w}_{im}^{(2)} \)
Matrix Factorization

Linear Network Hypothesis

'Linear Network' Hypothesis

\[ \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \]

\[ \mathbf{V}^T : \mathbf{w}^{(1)}_{ni} \]

\[ \mathbf{W} : \mathbf{w}^{(2)}_{im} \]

\[ \approx y_1 \]

\[ \approx y_2 \]

\[ \approx y_3 \]

\[ \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \]

\[ \{ (\mathbf{x}_n = \text{BinaryVectorEncoding}(n), \mathbf{y}_n = [r_{n1} \ ? \ ? \ r_{n4} \ r_{n5} \ldots \ r_{nM}]^T ) \} \]

- rename: \( \mathbf{V}^T \) for \( \mathbf{w}^{(1)}_{ni} \) and \( \mathbf{W} \) for \( \mathbf{w}^{(2)}_{im} \)

- hypothesis: \( h(\mathbf{x}) = \mathbf{x} \)
"Linear Network" Hypothesis

\[ \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \]

\[ \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \]

\[ \mathbf{V}^T : \mathbf{w}_{ni}^{(1)} \]

\[ \mathbf{W} : \mathbf{w}_{im}^{(2)} \]

\[ \{ (\mathbf{x}_n = \text{BinaryVectorEncoding}(n), \mathbf{y}_n = [r_{n1} \ ? \ ? \ r_{n4} \ r_{n5} \ \ldots \ r_{nM}]^T ) \} \]

- rename: \( \mathbf{V}^T \) for \( \mathbf{w}_{ni}^{(1)} \) and \( \mathbf{W} \) for \( \mathbf{w}_{im}^{(2)} \)

- hypothesis: \( \mathbf{h}(\mathbf{x}) = \mathbf{W}^T \mathbf{V} \mathbf{x} \)
‘Linear Network’ Hypothesis

\[ \mathbf{x} = \mathbf{V}^T : \mathbf{w}_{ni}^{(1)} \approx \mathbf{y}_1 \]
\[ \mathbf{W} : \mathbf{w}_{im}^{(2)} \approx \mathbf{y}_2 = \mathbf{y}_3 \]

\[ \left\{ (\mathbf{x}_n = \text{BinaryVectorEncoding}(n), \mathbf{y}_n = [r_{n1} \ ? \ ? \ r_{n4} \ r_{n5} \ \ldots \ r_{nM}]^T) \right\} \]

- rename: \( \mathbf{V}^T \) for \( \mathbf{w}_{ni}^{(1)} \) and \( \mathbf{W} \) for \( \mathbf{w}_{im}^{(2)} \)
- hypothesis: \( h(\mathbf{x}) = \mathbf{W}^T \mathbf{V} \mathbf{x} \)
- per-user output: \( h(\mathbf{x}_n) = \mathbf{W}^T \)
Matrix Factorization

‘Linear Network’ Hypothesis

\[ \mathbf{x} = \mathbf{V}^T \mathbf{w}_{ni}^{(1)} \quad \mathbf{W} : \mathbf{w}_{im}^{(2)} \approx \mathbf{y}_1 \approx \mathbf{y}_2 \approx \mathbf{y}_3 = \mathbf{y} \]

\[ \left\{ (\mathbf{x}_n = \text{BinaryVectorEncoding}(n), \mathbf{y}_n = [r_{n1}, r_{n4}, r_{n5}, \ldots, r_{nM}]^T) \right\} \]

- rename: \( \mathbf{V}^T \) for \( \mathbf{w}_{ni}^{(1)} \) and \( \mathbf{W} \) for \( \mathbf{w}_{im}^{(2)} \)
- hypothesis: \( \mathbf{h}(\mathbf{x}) = \mathbf{W}^T \mathbf{V} \mathbf{x} \)
- per-user output: \( \mathbf{h}(\mathbf{x}_n) = \mathbf{W}^T \mathbf{v}_n \), where \( \mathbf{v}_n \) is \( n \)-th column of \( \mathbf{V} \)
Matrix Factorization

Linear Network Hypothesis

\[ \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \quad \mathbf{v}^T : \mathbf{w}_{ni}^{(1)} \quad \mathbf{W} : \mathbf{w}_{im}^{(2)} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \]

\[ \{(\mathbf{x}_n = \text{BinaryVectorEncoding}(n), \mathbf{y}_n = [r_{n1} \ ? \ ? \ r_{n4} \ r_{n5} \ldots \ r_{nM}]^T)\} \]

- rename: \( \mathbf{V}^T \) for \( \mathbf{w}_{ni}^{(1)} \) and \( \mathbf{W} \) for \( \mathbf{w}_{im}^{(2)} \)
- hypothesis: \( \mathbf{h}(\mathbf{x}) = \mathbf{W}^T \mathbf{V} \mathbf{x} \)
- per-user output: \( \mathbf{h}(\mathbf{x}_n) = \mathbf{W}^T \mathbf{v}_n \), where \( \mathbf{v}_n \) is \( n \)-th column of \( \mathbf{V} \)

\textbf{linear network} for recommender system:

\textbf{learn} \( \mathbf{V} \) and \( \mathbf{W} \)
For $N$ users, $M$ movies, and $\tilde{d}$ ‘features’, how many variables need to be used to specify a linear network hypothesis $h(x) = W^T V x$?

1. $N + M + \tilde{d}$
2. $N \cdot M \cdot \tilde{d}$
3. $(N + M) \cdot \tilde{d}$
4. $(N \cdot M) + \tilde{d}$

Reference Answer: 3
For $N$ users, $M$ movies, and $	ilde{d}$ ‘features’, how many variables need to be used to specify a linear network hypothesis $h(x) = W^T V x$?

1. $N + M + \tilde{d}$
2. $N \cdot M \cdot \tilde{d}$
3. $(N + M) \cdot \tilde{d}$
4. $(N \cdot M) + \tilde{d}$

Reference Answer: 3

simply $N \cdot \tilde{d}$ for $V^T$ and $\tilde{d} \cdot M$ for $W$
Linear Network: Linear Model Per Movie

Linear network:

\[ h(x) = W^T \underbrace{\Phi(x)}_{Vx} \]
Matrix Factorization

Linear Network: Linear Model Per Movie

linear network:

\[ h(x) = W^T \underbrace{Vx}_{\Phi(x)} \]

—for \( m \)-th movie, just linear model \( h_m(x) = w^T_m \Phi(x) \)

subject to shared transform \( \Phi \)

---

Hsuan-Tien Lin  (NTU CSIE)  
Machine Learning Techniques
Linear Network: Linear Model Per Movie

linear network:

\[ h(x) = W^T \underbrace{\Phi(x)}_{\text{Vx}} \]

—for \( m \)-th movie, just linear model \( h_m(x) = w_m^T \Phi(x) \)

subject to shared transform \( \Phi \)

• for every \( D_m \), want \( r_{nm} = y_n \approx w_m^T v_n \)
Matrix Factorization

Basic Matrix Factorization

Linear Network: Linear Model Per Movie

linear network:

\[ h(x) = W^T \Phi(x) \]

— for \( m \)-th movie, just linear model \( h_m(x) = w_m^T \Phi(x) \)

subject to shared transform \( \Phi \)

- for every \( D_m \), want \( r_{nm} = y_n \approx w_m^T v_n \)
- \( E_{in} \) over all \( D_m \) with squared error measure:

\[
E_{in}(\{w_m\}, \{v_n\}) = \frac{1}{\sum_{m=1}^{M} |D_m|} \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - w_m^T v_n)^2
\]

Hsuan-Tien Lin (NTU CSIE)
Linear Network: Linear Model Per Movie

linear network:
\[ h(x) = W^T \Phi(x) \]

— for \( m \)-th movie, just linear model \( h_m(x) = w_m^T \Phi(x) \)
subject to shared transform \( \Phi \)

- for every \( D_m \), want \( r_{nm} = y_n \approx w_m^T v_n \)
- \( E_{in} \) over all \( D_m \) with squared error measure:

\[
E_{in}(\{w_m\}, \{v_n\}) = \frac{1}{\sum_{m=1}^{M} |D_m|} \sum_{\text{user } n \text{ rated movie } m} \left( r_{nm} - w_m^T v_n \right)^2
\]
Linear Network: Linear Model Per Movie

linear network:

\[ h(x) = W^T \Phi(x) \]

— for \( m \)-th movie, just linear model \( h_m(x) = W_m^T \Phi(x) \)

subject to shared transform \( \Phi \)

- for every \( D_m \), want \( r_{nm} = y_n \approx W_m^T v_n \)
- \( E_{in} \) over all \( D_m \) with squared error measure:

\[
E_{in}(\{w_m\}, \{v_n\}) = \frac{1}{\sum_{m=1}^M |D_m|} \sum_{\text{user } n \text{ rated movie } m} \left( r_{nm} - W_m^T v_n \right)^2
\]

linear network: transform and linear modelS jointly learned from all \( D_m \)
Matrix Factorization

\[ r_{nm} \approx w_m^T v_n = v_n^T w_m \]
## Basic Matrix Factorization

### Matrix Factorization

**Matrix Factorization**

Let $r_{nm}$ be the rating predicted for user $n$ on movie $m$. We can approximate this rating as:

$$ r_{nm} \approx w_m^T v_n = v_n^T w_m $$

**Match movie and viewer factors**

**Predicted rating**

- Comedy content
- Action content
- Blockbuster?
- Tom Cruise in it?
- Likes Tom Cruise?
- Prefers blockbusters?
- Likes action?
- Likes comedy?

**Movie and viewer contributions**

<table>
<thead>
<tr>
<th>$R$</th>
<th>movie$_1$</th>
<th>movie$_2$</th>
<th>...</th>
<th>movie$_M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>user$_1$</td>
<td>100</td>
<td>80</td>
<td>...</td>
<td>?</td>
</tr>
<tr>
<td>user$_2$</td>
<td>?</td>
<td>70</td>
<td>...</td>
<td>90</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>user$_N$</td>
<td>?</td>
<td>60</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

**Similar modeling** can be used for other abstract features.
Matrix Factorization

\[ r_{nm} \approx w_m^T v_n = v_n^T w_m \]

<table>
<thead>
<tr>
<th>R</th>
<th>movie(_1)</th>
<th>movie(_2)</th>
<th>...</th>
<th>movie(_M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>user(_1)</td>
<td>100</td>
<td>80</td>
<td>...</td>
<td>?</td>
</tr>
<tr>
<td>user(_2)</td>
<td>?</td>
<td>70</td>
<td>...</td>
<td>90</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>user(_N)</td>
<td>?</td>
<td>60</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \approx \]

\[ V^T \]

- \( v_1 \)
- \( v_2 \)
- ...
- \( v_N \)
Matrix Factorization

\[ r_{nm} \approx w_m^T v_n = v_n^T w_m \]

<table>
<thead>
<tr>
<th>R</th>
<th>movie_1</th>
<th>movie_2</th>
<th>\cdots</th>
<th>movie_M</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_1</td>
<td>100</td>
<td>80</td>
<td>\cdots</td>
<td>?</td>
</tr>
<tr>
<td>user_2</td>
<td>?</td>
<td>70</td>
<td>\cdots</td>
<td>90</td>
</tr>
<tr>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
</tr>
<tr>
<td>user_N</td>
<td>?</td>
<td>60</td>
<td>\cdots</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \approx \]

\[ V^T \]

\[ W \]

\[ w_1 \]

\[ w_2 \]

\[ \cdots \]

\[ w_M \]

Match movie and viewer factors

predicted rating


Add contributions from each factor.
Matrix Factorization

\[ r_{nm} \approx w_m^T v_n = v_n^T w_m \iff R \approx V^T W \]

<table>
<thead>
<tr>
<th>R</th>
<th>movie_1</th>
<th>movie_2</th>
<th>\cdots</th>
<th>movie_M</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_1</td>
<td>100</td>
<td>80</td>
<td>\cdots</td>
<td>?</td>
</tr>
<tr>
<td>user_2</td>
<td>?</td>
<td>70</td>
<td>\cdots</td>
<td>90</td>
</tr>
<tr>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
<td>\cdots</td>
</tr>
<tr>
<td>user_N</td>
<td>?</td>
<td>60</td>
<td>\cdots</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ V^T \]

\[ W \]

\[ w_1 \]

\[ w_2 \]

\[ \cdots \]

\[ w_M \]
Matrix Factorization

\[ r_{nm} \approx w_m^T v_n = v_n^T w_m \iff R \approx V^T W \]

\[ \begin{array}{c|cccc}
\hline
 & \text{movie}_1 & \text{movie}_2 & \cdots & \text{movie}_M \\
\hline
\text{user}_1 & 100 & 80 & \cdots & ? \\
\text{user}_2 & ? & 70 & \cdots & 90 \\
\ldots & \ldots & \ldots & \cdots & \ldots \\
\text{user}_N & ? & 60 & \cdots & 0 \\
\hline
\end{array} \]

Matrix Factorization Model

learning:
- known rating
  \[ \rightarrow \text{learned factors } v_n \text{ and } w_m \]
- unknown rating prediction

Match movie and viewer factors
add contributions from each factor
predicted rating
Matrix Factorization

\[ r_{nm} \approx w_m^T v_n = v_n^T w_m \quad \iff \quad R \approx V^T W \]

### Matrix Factorization Model

**learning:**

- known rating
  \[ \rightarrow \text{learned factors } v_n \text{ and } w_m \]
- unknown rating prediction

**predicted rating**

**add contributions from each factor**

**similar modeling can be used for other abstract features**

---

Hsuan-Tien Lin (NTU CSIE)

Machine Learning Techniques
Matrix Factorization Learning

\[
\min_{\mathbf{W}, \mathbf{V}} E_{\text{in}}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2
\]
Matrix Factorization Learning

\[
\min_{W,V} E_{\text{in}}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} \left( r_{nm} - w_m^T v_n \right)^2
\]

\[
= \sum_{m=1}^{M} \left( \sum_{(x_n, r_{nm}) \in D_m} \left( r_{nm} - w_m^T v_n \right)^2 \right)
\]
Matrix Factorization Learning

\[
\min_{W,V} E_{\text{in}}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} \left( r_{nm} - w_m^T v_n \right)^2 \\
= \sum_{m=1}^{M} \left( \sum_{(x_n,r_{nm}) \in \mathcal{D}_m} \left( r_{nm} - w_m^T v_n \right)^2 \right)
\]

- two sets of variables:
Matrix Factorization Learning

\[
\min_{W,V} E_{\text{in}}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - w_m^T v_n)^2
\]

\[
= \sum_{m=1}^{M} \left( \sum_{(x_n,r_{nm}) \in D_m} (r_{nm} - w_m^T v_n)^2 \right)
\]

- two sets of variables:
  - can consider alternating minimization, remember? :-)

Hsuan-Tien Lin  (NTU CSIE)  Machine Learning Techniques  9/22
Matrix Factorization Learning

\[
\min_{W,V} E_{in}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - w_m^T v_n)^2
\]

\[
= \sum_{m=1}^M \left( \sum_{(x_n, r_{nm}) \in D_m} (r_{nm} - w_m^T v_n)^2 \right)
\]

- two sets of variables:
  - can consider **alternating minimization**, remember? :-)  
- when \(v_n\) fixed, minimizing \(w_m\)
Matrix Factorization Learning

\[
\min_{W, V} E_{\text{in}}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} \left( r_{nm} - w_m^T v_n \right)^2
\]

\[
= \sum_{m=1}^{M} \left( \sum_{(x_n, r_{nm}) \in \mathcal{D}_m} \left( r_{nm} - w_m^T v_n \right)^2 \right)
\]

- two sets of variables: can consider **alternating minimization, remember? :-)**
- when \( v_n \) fixed, minimizing \( w_m \) \( \equiv \) minimize \( E_{\text{in}} \) within \( \mathcal{D}_m \)
Matrix Factorization Learning

\[
\min_{\mathbf{W}, \mathbf{V}} E_{\text{in}}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2 \\
= \sum_{m=1}^{M} \left( \sum_{(x_n, r_{nm}) \in \mathcal{D}_m} (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n)^2 \right)
\]

- two sets of variables:
  can consider alternating minimization, remember? :-)
- when \( \mathbf{v}_n \) fixed, minimizing \( \mathbf{w}_m \equiv \) minimize \( E_{\text{in}} \) within \( \mathcal{D}_m \)
  —simply per-movie (per-\( \mathcal{D}_m \)) linear regression without \( w_0 \)
Matrix Factorization Learning

\[
\min_{W, V} E_{\text{in}}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - w_m^T v_n)^2
\]
\[
= \sum_{m=1}^{M} \left( \sum_{(x_n, r_{nm}) \in D_m} (r_{nm} - w_m^T v_n)^2 \right)
\]

- two sets of variables: can consider alternating minimization, remember? :-)
- when \( v_n \) fixed, minimizing \( w_m \equiv \min E_{\text{in}} \) within \( D_m \) —simply per-movie (per-\( D_m \)) linear regression without \( w_0 \)
- when \( w_m \) fixed, minimizing \( v_n \)?
Matrix Factorization Learning

\[
\min_{W, V} E_{\text{in}}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - w_m^T v_n)^2
\]

\[
= \sum_{m=1}^{M} \left( \sum_{(x_n, r_{nm}) \in D_m} (r_{nm} - w_m^T v_n)^2 \right)
\]

- two sets of variables: can consider alternating minimization, remember? :-)
- when \( v_n \) fixed, minimizing \( w_m \) \( \equiv \) minimize \( E_{\text{in}} \) within \( D_m \) —simply per-movie (per-\( D_m \)) linear regression without \( w_0 \)
- when \( w_m \) fixed, minimizing \( v_n \)?

by symmetry between users/movies
Matrix Factorization Learning

\[
\min_{W,V} E_{in}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - w^T_m v_n)^2 = \sum_{m=1}^M \left( \sum_{(x_n, r_{nm}) \in D_m} (r_{nm} - w^T_m v_n)^2 \right)
\]

- two sets of variables: can consider \textbf{alternating minimization, remember? :-)}
- when \(v_n\) fixed, minimizing \(w_m\) \(\equiv\) minimize \(E_{in}\) within \(D_m\) —simply per-movie (per-\(D_m\)) \textbf{linear regression} without \(w_0\)
- when \(w_m\) fixed, minimizing \(v_n\)? —per-user linear regression \textbf{without} \(v_0\) by \textbf{symmetry} between users/movies
Matrix Factorization Learning

$$\min_{W,V} E_{in}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - w_m^T v_n)^2$$

$$= \sum_{m=1}^{M} \left( \sum_{(x_n,r_{nm}) \in D_m} (r_{nm} - w_m^T v_n)^2 \right)$$

- **two sets** of variables:
  - can consider **alternating minimization**, remember? :-)
- when \(v_n\) fixed, minimizing \(w_m\) \(\equiv\) minimize \(E_{in}\) within \(D_m\)
  —simply per-movie (per-\(D_m\)) **linear regression** without \(w_0\)
- when \(w_m\) fixed, minimizing \(v_n\)?
  —per-user linear regression without \(v_0\)
  by symmetry between users/movies

called **alternating least squares** algorithm
Alternating Least Squares

2. **Alternating optimization** of $E_{in}$: repeatedly

until converge
Alternating Least Squares

- **alternating optimization** of $E_{in}$: repeatedly
  1. optimize $w_1, w_2, \ldots, w_M$:
     update $w_m$ by *m*-th-movie linear regression on $\{(v_n, r_{nm})\}$

until **converge**
Alternating Least Squares

Alternating optimization of $E_{in}$: repeatedly

1. optimize $w_1, w_2, \ldots, w_M$:
   update $w_m$ by $m$-th-movie linear regression on $\{(v_n, r_{nm})\}$

2. optimize $v_1, v_2, \ldots, v_N$:
   update $v_n$ by $n$-th-user linear regression on $\{(w_m, r_{nm})\}$

until converge
Alternating Least Squares

1. **initialize** $\tilde{d}$ dimension vectors $\{w_m\}$, $\{v_n\}$

2. **alternating optimization** of $E_{in}$: repeatedly
   - optimize $w_1, w_2, \ldots, w_M$:
     - update $w_m$ by $m$-th-movie linear regression on $\{(v_n, r_{nm})\}$
   - optimize $v_1, v_2, \ldots, v_N$:
     - update $v_n$ by $n$-th-user linear regression on $\{(w_m, r_{nm})\}$

until converge

- **initialize**: usually just randomly
Alternating Least Squares

initialize \( \tilde{d} \) dimension vectors \( \{w_m\}, \{v_n\} \)

alternating optimization of \( E_{in} \): repeatedly

1. optimize \( w_1, w_2, \ldots, w_M \):
   update \( w_m \) by \( m \)-th-movie linear regression on \( \{(v_n, r_{nm})\} \)

2. optimize \( v_1, v_2, \ldots, v_N \):
   update \( v_n \) by \( n \)-th-user linear regression on \( \{(w_m, r_{nm})\} \)

until converge

- initialize: usually just randomly
- converge: guaranteed as \( E_{in} \) decreases during alternating minimization
Alternating Least Squares

1. initialize \( \tilde{d} \) dimension vectors \( \{w_m\}, \{v_n\} \)
2. **alternating optimization** of \( E_{\text{in}} \): repeatedly
   1. optimize \( w_1, w_2, \ldots, w_M \):
      update \( w_m \) by \( m\)-th-movie linear regression on \( \{(v_n, r_{nm})\} \)
   2. optimize \( v_1, v_2, \ldots, v_N \):
      update \( v_n \) by \( n\)-th-user linear regression on \( \{(w_m, r_{nm})\} \)

until converge

- **initialize**: usually just randomly
- **converge**: guaranteed as \( E_{\text{in}} \) decreases during alternating minimization

alternating least squares: the ‘tango’ dance between users/movies
Linear Autoencoder versus Matrix Factorization

**Matrix Factorization**

\[ R \approx V^T W \]
Linear Autoencoder versus Matrix Factorization

**Linear Autoencoder**

\[ X \approx W (W^T X) \]

**Matrix Factorization**

\[ R \approx V^T W \]

- **Motivation:** Special - ~d linear NNet
- **Error Measure:** Squared on all \( x_{ni} \)
- **Solution:** Global optimal at eigenvectors of \( X^T X \)
- **Usefulness:** Extract dimension-reduced features

**Linear Autoencoder** is equivalent to a special matrix factorization of the complete data matrix.
## Linear Autoencoder versus Matrix Factorization

### Linear Autoencoder

\[ X \approx W (W^T X) \]

- **motivation:**
  - special \(d \sim d\)-linear NNet

### Matrix Factorization

\[ R \approx V^T W \]

- **solution:** local optimal via alternating least squares

- **usefulness:** extract hidden user/movie features

---

Hsuan-Tien Lin (NTU CSIE)

Machine Learning Techniques
## Linear Autoencoder versus Matrix Factorization

<table>
<thead>
<tr>
<th><strong>Linear Autoencoder</strong></th>
<th><strong>Matrix Factorization</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \approx W (W^T X)$</td>
<td>$R \approx V^T W$</td>
</tr>
<tr>
<td>• motivation:</td>
<td>• motivation:</td>
</tr>
<tr>
<td>special $d$-$\tilde{d}$-$d$ linear NNet</td>
<td>$N$-$\tilde{d}$-$M$ linear NNet</td>
</tr>
</tbody>
</table>
Matrix Factorization

Basic Matrix Factorization

Linear Autoencoder versus Matrix Factorization

**Linear Autoencoder**

\[ X \approx W (W^T X) \]

- motivation: special $d$-$\tilde{d}$-$d$ linear NNet
- error measure: squared on all $x_{ni}$

**Matrix Factorization**

\[ R \approx V^T W \]

- motivation: $N$-$\tilde{d}$-$M$ linear NNet
### Linear Autoencoder versus Matrix Factorization

#### Linear Autoencoder

\[ X \approx W (W^T X) \]

- **motivation:** special \( d \)-\( \tilde{d} \)-\( d \) linear NNet
- **error measure:** squared on all \( x_{ni} \)

#### Matrix Factorization

\[ R \approx V^T W \]

- **motivation:** \( N \)-\( \tilde{d} \)-\( M \) linear NNet
- **error measure:** squared on known \( r_{nm} \)
## Linear Autoencoder versus Matrix Factorization

### Linear Autoencoder

\[ X \approx W (W^T X) \]

- motivation: special \( d - \tilde{d} - d \) linear NNet
- error measure: squared on all \( x_{ni} \)
- solution: global optimal at eigenvectors of \( X^T X \)

### Matrix Factorization

\[ R \approx V^T W \]

- motivation: \( N - \tilde{d} - M \) linear NNet
- error measure: squared on known \( r_{nm} \)
### Linear Autoencoder versus Matrix Factorization

**Linear Autoencoder**

\[ X \approx W (W^T X) \]

- **motivation:** special \( d \)-\( \tilde{d} \)-\( d \) linear NNet
- **error measure:** squared on all \( x_{ni} \)
- **solution:** global optimal at eigenvectors of \( X^T X \)

**Matrix Factorization**

\[ R \approx V^T W \]

- **motivation:** \( N \)-\( \tilde{d} \)-\( M \) linear NNet
- **error measure:** squared on known \( r_{nm} \)
- **solution:** local optimal via alternating least squares

---

**Motivation:**

- **Linear Autoencoder**
  - Extracts dimension-reduced features
- **Matrix Factorization**
  - Extracts hidden user/movie features
# Linear Autoencoder versus Matrix Factorization

## Linear Autoencoder

\[ \mathbf{X} \approx \mathbf{W} (\mathbf{W}^T \mathbf{X}) \]

- **motivation:** special \(d\)-\(d\) linear NNet
- **error measure:** squared on all \(x_{ni}\)
- **solution:** global optimal at eigenvectors of \(\mathbf{X}^T \mathbf{X}\)
- **usefulness:** extract dimension-reduced features

## Matrix Factorization

\[ \mathbf{R} \approx \mathbf{V}^T \mathbf{W} \]

- **motivation:** \(N\)-\(\tilde{d}\)-\(M\) linear NNet
- **error measure:** squared on known \(r_{nm}\)
- **solution:** local optimal via alternating least squares
Linear Autoencoder versus Matrix Factorization

**Linear Autoencoder**

\[ X \approx W (W^T X) \]

- motivation: special \(d\)-\(\tilde{d}\)-\(d\) linear NNet
- error measure: squared on all \(x_{ni}\)
- solution: global optimal at eigenvectors of \(X^T X\)
- usefulness: extract dimension-reduced features

**Matrix Factorization**

\[ R \approx V^T W \]

- motivation: \(N\)-\(\tilde{d}\)-\(M\) linear NNet
- error measure: squared on known \(r_{nm}\)
- solution: local optimal via alternating least squares
- usefulness: extract hidden user/movie features
Linear Autoencoder versus Matrix Factorization

**Linear Autoencoder**

\[ X \approx W (W^T X) \]

- motivation: special \( d \rightarrow \tilde{d} \rightarrow d \) linear NNet
- error measure: squared on all \( x_{ni} \)
- solution: global optimal at eigenvectors of \( X^T X \)
- usefulness: extract dimension-reduced features

**Matrix Factorization**

\[ R \approx V^T W \]

- motivation: \( N \rightarrow \tilde{d} \rightarrow M \) linear NNet
- error measure: squared on known \( r_{nm} \)
- solution: local optimal via alternating least squares
- usefulness: extract hidden user/movie features

linear autoencoder \( \equiv \text{special} \) matrix factorization of complete \( X \)
How many least squares problems does the alternating least squares algorithm needs to solve in one iteration of alternation?

1. number of movies $M$
2. number of users $N$
3. $M + N$
4. $M \cdot N$
How many least squares problems does the alternating least squares algorithm needs to solve in one iteration of alternation?

1. number of movies $M$
2. number of users $N$
3. $M + N$
4. $M \cdot N$

Reference Answer: 3

simply $M$ per-movie problems and $N$ per-user problems
Another Possibility: Stochastic Gradient Descent

$$E_{in}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} \left( r_{nm} - w_m^T v_n \right)^2$$
Another Possibility: Stochastic Gradient Descent

\[ E_{\text{in}}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - w_m^T v_n)^2 \]

SGD: randomly pick **one example** within the \( \sum \) & update with **gradient to per-example** err, remember? :-)
Another Possibility: Stochastic Gradient Descent

\[ E_{\text{in}}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} \left( r_{nm} - w_m^T v_n \right)^2 \]

SGD: randomly pick one example within the sum & update with gradient to per-example error, remember? :-)

- ‘efficient’ per iteration
Another Possibility: Stochastic Gradient Descent

\[ E_{in}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - w_m^T v_n)^2 \]

SGD: randomly pick one example within the sum & update with gradient to per-example err, remember? :-)

- ‘efficient’ per iteration
- simple to implement
Another Possibility: Stochastic Gradient Descent

\[ E_{in}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} \left( r_{nm} - w_m^T v_n \right)^2 \]

SGD: randomly pick one example within the \( \sum \) & update with gradient to per-example err, remember? :-)

- ‘efficient’ per iteration
- simple to implement
- easily extends to other err
Another Possibility: Stochastic Gradient Descent

\[ E_{in}(\{w_m\}, \{v_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - w_m^T v_n)^2 \]

SGD: randomly pick one example within the \( \sum \) & update with gradient to per-example error, remember? :-)

- ‘efficient’ per iteration
- simple to implement
- easily extends to other error

next: SGD for matrix factorization
Gradient of Per-Example Error Function

\[ \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( \frac{r_{nm} - w^T_m v_n}{2} \right)^2 \]
Gradient of Per-Example Error Function

$$\text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( r_{nm} - w_m^T v_n \right)^2$$
Gradient of Per-Example Error Function

\[
\text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( r_{nm} - w_m^T v_n \right)^2
\]

\[
\nabla_{v_{1126}} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) =
\]
Gradient of Per-Example Error Function

\[ \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( r_{nm} - w_m^T v_n \right)^2 \]

\[ \nabla_{v_{1126}} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } n = 1126 \]
Matrix Factorization

**Stochastic Gradient Descent**

Gradient of Per-Example Error Function

$$\text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( r_{nm} - w_m^T v_n \right)^2$$

\[
\nabla_{v_{1126}} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } n = 1126
\]

\[
\nabla_{w_{6211}} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } m =
\]
Gradient of Per-Example Error Function

\[ \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( r_{nm} - w_m^T v_n \right)^2 \]

\[ \nabla v_{1126} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } n = 1126 \]

\[ \nabla w_{6211} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } m = 6211 \]
Gradient of Per-Example Error Function

\[
\text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( r_{nm} - w_m^T v_n \right)^2
\]

\[
\nabla_{v_{1126}} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } n = 1126
\]

\[
\nabla_{w_{6211}} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } m = 6211
\]

\[
\nabla_{v_n} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - w_m^T v_n \right) w_m^T
\]
Matrix Factorization

Stochastic Gradient Descent

Gradient of Per-Example Error Function

\[
\text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( r_{nm} - w_m^T v_n \right)^2
\]

\[
\nabla v_{1126} \quad \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } n = 1126
\]

\[
\nabla w_{6211} \quad \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } m = 6211
\]

\[
\nabla v_n \quad \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - w_m^T v_n \right) w_m
\]
Gradient of Per-Example Error Function

\[ \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( r_{nm} - \mathbf{w}_m^T \mathbf{v}_n \right)^2 \]

\[ \nabla_{\mathbf{v}_{1126}} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } n = 1126 \]
\[ \nabla_{\mathbf{w}_{6211}} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } m = 6211 \]
\[ \nabla_{\mathbf{v}_n} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - \mathbf{w}_m^T \mathbf{v}_n \right) \mathbf{w}_m \]
\[ \nabla_{\mathbf{w}_m} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - \mathbf{w}_m^T \mathbf{v}_n \right) \]
Gradient of Per-Example Error Function

\[ \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( r_{nm} - w_m^T v_n \right)^2 \]

\[ \nabla_{v_{1126}} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } n = 1126 \]

\[ \nabla_{v_{6211}} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } m = 6211 \]

\[ \nabla_{v_n} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - w_m^T v_n \right) w_m \]

\[ \nabla_{w_m} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - w_m^T v_n \right) v_n \]
Gradient of Per-Example Error Function

\[ \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( r_{nm} - w_m^T v_n \right)^2 \]

\[ \nabla_{v_{1126}} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } n = 1126 \]
\[ \nabla_{w_{6211}} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } m = 6211 \]
\[ \nabla_{v_n} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - w_m^T v_n \right) w_m \]
\[ \nabla_{w_m} \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - w_m^T v_n \right) v_n \]

per-example gradient
\[ \propto - ( \text{residual} ) ( \text{the other feature vector} ) \]
Gradient of Per-Example Error Function

\[
\text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = \left( r_{nm} - w_m^T v_n \right)^2
\]

\[
\nabla v_{1126} \quad \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } n = 1126
\]

\[
\nabla w_{6211} \quad \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } m = 6211
\]

\[
\nabla v_n \quad \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - w_m^T v_n \right) w_m
\]

\[
\nabla w_m \quad \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - w_m^T v_n \right) v_n
\]

per-example gradient

\[
\propto -\text{(residual)}(\quad )
\]
Gradient of Per-Example Error Function

\[ \text{err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = (r_{nm} - w_m^T \mathbf{v}_n)^2 \]

\[ \nabla \mathbf{v}_{126} \text{ err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } n = 1126 \]

\[ \nabla w_{6211} \text{ err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = 0 \text{ unless } m = 6211 \]

\[ \nabla \mathbf{v}_n \text{ err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - w_m^T \mathbf{v}_n \right) w_m \]

\[ \nabla w_m \text{ err}(\text{user } n, \text{ movie } m, \text{ rating } r_{nm}) = -2 \left( r_{nm} - w_m^T \mathbf{v}_n \right) \mathbf{v}_n \]

per-example gradient

\[ \propto -\text{(residual)} \text{(the other feature vector)} \]
for $t = 0, 1, \ldots, T$
SGD for Matrix Factorization

for $t = 0, 1, \ldots, T$

1. randomly pick $(n, m)$ within all known $r_{nm}$
Matrix Factorization
Stochastic Gradient Descent

SGD for Matrix Factorization

for $t = 0, 1, \ldots, T$

1. randomly pick $(n, m)$ within all known $r_{nm}$
2. calculate residual $\tilde{r}_{nm} = (r_{nm} - w_m^T v_n)$
for $t = 0, 1, \ldots, T$

1. randomly pick $(n, m)$ within all known $r_{nm}$
2. calculate residual $\tilde{r}_{nm} = (r_{nm} - w_m^T v_n)$
3. SGD-update:

$$v_n^{new} \leftarrow v_n^{old} + \eta \cdot \tilde{r}_{nm}.$$
SGD for Matrix Factorization

for $t = 0, 1, \ldots, T$

1. randomly pick $(n, m)$ within all known $r_{nm}$
2. calculate residual $\tilde{r}_{nm} = (r_{nm} - w_m^T v_n)$
3. SGD-update:

$$v_n^{\text{new}} \leftarrow v_n^{\text{old}} + \eta \cdot \tilde{r}_{nm} w_m^{\text{old}}$$
for $t = 0, 1, \ldots, T$

1. randomly pick $(n, m)$ within all known $r_{nm}$
2. calculate residual $\tilde{r}_{nm} = (r_{nm} - w_m^T v_n)$
3. SGD-update:

\[
\begin{align*}
\mathbf{v}_n^{\text{new}} &\leftarrow \mathbf{v}_n^{\text{old}} + \eta \cdot \tilde{r}_{nm} \mathbf{w}_m^{\text{old}} \\
\mathbf{w}_m^{\text{new}} &\leftarrow \mathbf{w}_m^{\text{old}} + \eta \cdot \tilde{r}_{nm}
\end{align*}
\]
for $t = 0, 1, \ldots, T$

1. randomly pick $(n, m)$ within all known $r_{nm}$
2. calculate residual $\tilde{r}_{nm} = (r_{nm} - w_m^T v_n)$
3. SGD-update:

$$\begin{align*}
    v_n^{\text{new}} &\leftarrow v_n^{\text{old}} + \eta \cdot \tilde{r}_{nm} w_m^{\text{old}} \\
    w_m^{\text{new}} &\leftarrow w_m^{\text{old}} + \eta \cdot \tilde{r}_{nm} v_n^{\text{old}}
\end{align*}$$
SGD for Matrix Factorization

initialize \( \tilde{d} \) dimension vectors \( \{ \mathbf{w}_m \}, \{ \mathbf{v}_n \} \) randomly
for \( t = 0, 1, \ldots, T \)

1. randomly pick \((n, m)\) within all known \( r_{nm} \)
2. calculate residual \( \tilde{r}_{nm} = (r_{nm} - \mathbf{w}_m^T \mathbf{v}_n) \)
3. SGD-update:

\[
\begin{align*}
\mathbf{v}_n^{\text{new}} &\leftarrow \mathbf{v}_n^{\text{old}} + \eta \cdot \tilde{r}_{nm} \mathbf{w}_m^{\text{old}} \\
\mathbf{w}_m^{\text{new}} &\leftarrow \mathbf{w}_m^{\text{old}} + \eta \cdot \tilde{r}_{nm} \mathbf{v}_n^{\text{old}}
\end{align*}
\]
Matrix Factorization

Stochastic Gradient Descent

SGD for Matrix Factorization

initialize $\tilde{d}$ dimension vectors $\{w_m\}, \{v_n\}$ randomly
for $t = 0, 1, \ldots, T$

1. randomly pick $(n, m)$ within all known $r_{nm}$
2. calculate residual $\tilde{r}_{nm} = (r_{nm} - w_m^T v_n)$
3. SGD-update:

$$v_n^{new} \leftarrow v_n^{old} + \eta \cdot \tilde{r}_{nm} w_m^{old}$$
$$w_m^{new} \leftarrow w_m^{old} + \eta \cdot \tilde{r}_{nm} v_n^{old}$$

SGD: perhaps most popular large-scale matrix factorization algorithm
Matrix Factorization

Stochastic Gradient Descent

SGD for Matrix Factorization in Practice

KDDCup 2011 Track 1: World Champion Solution by NTU

specialty of data (application need):
per-user training ratings earlier than test ratings in time

training/test mismatch: typical sampling bias, remember? :-)

want: emphasize latter examples

last $T'$ iterations of SGD: only those $T'$ examples considered - learned $\{w_m\}, \{v_n\}$ favoring those

our idea: time-deterministic SGD that visits latter examples last - consistent improvements of test performance

if you understand the behavior of techniques, easier to modify for your real-world use
KDDCup 2011 Track 1: World Champion Solution by NTU

- specialty of data (application need):
KDDCup 2011 Track 1: World Champion Solution by NTU

- specialty of data (application need):
  per-user training ratings **earlier than** test ratings **in time**
SGD for Matrix Factorization in Practice

KDDCup 2011 Track 1: World Champion Solution by NTU

- specialty of data (application need): per-user training ratings **earlier than** test ratings **in time**
- training/test mismatch: typical **sampling bias, remember? :-)**
KDDCup 2011 Track 1: World Champion Solution by NTU

- specialty of data (application need): per-user training ratings earlier than test ratings in time
- training/test mismatch: typical sampling bias, remember? :-)

- want: emphasize latter examples
KDDCup 2011 Track 1: World Champion Solution by NTU

- specialty of data (application need): per-user training ratings earlier than test ratings in time
- training/test mismatch: typical sampling bias, remember? :-)

- want: emphasize latter examples
- last $T'$ iterations of SGD: only those $T'$ examples considered
KDDCup 2011 Track 1: World Champion Solution by NTU

- specialty of data (application need): per-user training ratings earlier than test ratings in time
- training/test mismatch: typical sampling bias, remember? :-)

- want: emphasize latter examples
- last $T'$ iterations of SGD: only those $T'$ examples considered—learned $\{w_m\}, \{v_n\}$ favoring those
SGD for Matrix Factorization in Practice

KDDCup 2011 Track 1: World Champion Solution by NTU

- specialty of data (application need): per-user training ratings earlier than test ratings in time
- training/test mismatch: typical sampling bias, remember? :-)

- want: emphasize latter examples
- last $T'$ iterations of SGD: only those $T'$ examples considered —learned $\{w_m\}, \{v_n\}$ favoring those
- our idea: time-deterministic SGD that visits latter examples last
SGD for Matrix Factorization in Practice

KDDCup 2011 Track 1: World Champion Solution by NTU

- specialty of data (application need): per-user training ratings earlier than test ratings in time
- training/test mismatch: typical sampling bias, remember? :-)

- want: emphasize latter examples
- last $T'$ iterations of SGD: only those $T'$ examples considered —learned $\{w_m\}, \{v_n\}$ favoring those
- our idea: time-deterministic SGD that visits latter examples last —consistent improvements of test performance
SGD for Matrix Factorization in Practice

KDDCup 2011 Track 1: World Champion Solution by NTU

- specialty of data (application need): per-user training ratings earlier than test ratings in time
- training/test mismatch: typical sampling bias, remember? :-)

- want: emphasize latter examples
- last $T'$ iterations of SGD: only those $T'$ examples considered —learned $\{w_m\}$, $\{v_n\}$ favoring those
- our idea: time-deterministic SGD that visits latter examples last —consistent improvements of test performance

if you understand the behavior of techniques, easier to modify for your real-world use
If all \( w_m \) and \( v_n \) are initialized to the 0 vector, what will NOT happen in SGD for matrix factorization?

1. all \( w_m \) are always 0
2. all \( v_n \) are always 0
3. every residual \( \tilde{r}_{nm} = \) the original rating \( r_{nm} \)
4. \( E_{in} \) decreases after each SGD update

Reference Answer:

The 0 feature vectors provides a per-example gradient of 0 for every example. So \( E_{in} \) cannot be further decreased.
If all $w_m$ and $v_n$ are initialized to the $0$ vector, what will NOT happen in SGD for matrix factorization?

1. All $w_m$ are always $0$
2. All $v_n$ are always $0$
3. Every residual $\tilde{r}_{nm} = \text{the original rating } r_{nm}$
4. $E_{in}$ decreases after each SGD update

**Reference Answer:** 4

The $0$ feature vectors provides a per-example gradient of $0$ for every example. So $E_{in}$ cannot be further decreased.
extraction models: feature transform $\Phi$ as hidden variables in addition to linear model
Map of Extraction Models

**extraction models:** feature transform $\Phi$ as hidden variables in addition to linear model
Map of Extraction Models

**extraction models:** feature transform $\Phi$ as hidden variables in addition to linear model

**Neural Network/Deep Learning**
- weights $w_{ij}^{(\ell)}$
- weights $w_{ij}^{(L)}$

**RBF Network**
- RBF centers $\mu_m$
- weights $\beta_m$
Matrix Factorization

Summary of Extraction Models

Map of Extraction Models

**extraction models:** feature transform $\Phi$ as hidden variables in addition to linear model

---

**Neural Network/Deep Learning**
- weights $w_{ij}^{(\ell)}$
- weights $w_{ij}^{(L)}$

**RBF Network**
- RBF centers $\mu_m$
- weights $\beta_m$

**Matrix Factorization**
- user features $v_n$
- movie features $w_m$
Matrix Factorization

Summary of Extraction Models

Map of Extraction Models

*extraction models*: feature transform $\Phi$ as hidden variables in addition to linear model

Adaptive/Gradient Boosting

hypotheses $g_t$; weights $\alpha_t$

Neural Network/Deep Learning

weights $w_{ij}^{(\ell)}$; weights $w_{ij}^{(L)}$

RBF Network

RBF centers $\mu_m$; weights $\beta_m$

Matrix Factorization

user features $v_n$; movie features $w_m$
**Matrix Factorization**

**Summary of Extraction Models**

**Map of Extraction Models**

*extraction models: feature transform $\Phi$ as hidden variables in addition to linear model*

- **Adaptive/Gradient Boosting**
  - hypotheses $g_t$; weights $\alpha_t$

- **Neural Network/Deep Learning**
  - weights $w_{ij}^{(\ell)}$; weights $w_{ij}^{(L)}$

- **RBF Network**
  - RBF centers $\mu_m$; weights $\beta_m$

- **Matrix Factorization**
  - user features $\mathbf{v}_n$; movie features $\mathbf{w}_m$

- **k Nearest Neighbor**
  - $x_n$-neighbor RBF; weights $y_n$
Map of Extraction Models

**extraction models:** feature transform $\Phi$ as hidden variables in addition to linear model

Adaptive/Gradient Boosting
- hypotheses $g_t$; weights $\alpha_t$

Neural Network/Deep Learning
- weights $w_{ij}^{(\ell)}$
- weights $w_{ij}^{(L)}$

RBF Network
- RBF centers $\mu_m$
- weights $\beta_m$

Matrix Factorization
- user features $v_n$
- movie features $w_m$

$k$ Nearest Neighbor
- $x_n$-neighbor RBF
- weights $y_n$

extraction models: a rich family
Matrix Factorization

Summary of Extraction Models

Map of Extraction Techniques

Adaptive/Gradient Boosting

functional gradient descent
Matrix Factorization

Summary of Extraction Models

Map of Extraction Techniques

Adaptive/Gradient Boosting
- functional gradient descent

Neural Network/Deep Learning
- SGD (backprop)
Map of Extraction Techniques

Adaptive/Gradient Boosting
- functional gradient descent

Neural Network/Deep Learning
- SGD (backprop)
- autoencoder
Map of Extraction Techniques

Adaptive/Gradient Boosting
- functional gradient descent

Neural Network/Deep Learning
- SGD (backprop)
- autoencoder

RBF Network
- $k$-means clustering
Map of Extraction Techniques

- Adaptive/Gradient Boosting
  - functional gradient descent

- Neural Network/Deep Learning
  - SGD (backprop)
  - autoencoder

- RBF Network
  - $k$-means clustering

- Matrix Factorization
  - SGD
  - alternating leastSQR
Adaptive/Gradient Boosting
- functional gradient descent

Neural Network/Deep Learning
- SGD (backprop)
- autoencoder

RBF Network
- k-means clustering

Matrix Factorization
- SGD
- alternating leastSQR

k Nearest Neighbor
- lazy learning :-)

Matrix Factorization
Summary of Extraction Models
Map of Extraction Techniques
Map of Extraction Techniques

Adaptive/Gradient Boosting
- functional gradient descent

Neural Network/Deep Learning
- SGD (backprop)
- autoencoder

RBF Network
- k-means clustering

Matrix Factorization
- SGD
- alternating leastSQR

k Nearest Neighbor
- lazy learning :-)

extraction techniques: quite diverse
## Pros and Cons of Extraction Models

<table>
<thead>
<tr>
<th>Extraction Models</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Network/Deep Learning</td>
<td>• Easy: reduces human burden in designing features</td>
<td>• Hard: non-convex optimization problems in general</td>
</tr>
<tr>
<td>RBF Network</td>
<td>• Powerful: if enough hidden variables considered</td>
<td>• Overfitting: needs proper regularization/validation</td>
</tr>
<tr>
<td>Matrix Factorization</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Pros:**
- Easy: reduces human burden in designing features.
- Powerful: if enough hidden variables considered.

**Cons:**
- Hard: non-convex optimization problems in general.
- Overfitting: needs proper regularization/validation.
Pros and Cons of Extraction Models

- **Pros**
  - ‘easy’: reduces human burden in designing features

- **Cons**
  - **hard**: non-convex optimization problems in general
  - overfitting: needs proper regularization/validation

Extraction Models:
- Neural Network/Deep Learning
- RBF Network
- Matrix Factorization
Pros and Cons of Extraction Models

**Pros**
- *'easy':*
  reduces **human burden** in designing features

**Cons**
- *'hard':*
  **non-convex** optimization problems in general
Pros and Cons of Extraction Models

**Pros**
- `easy`: reduces human burden in designing features
- `powerful`: if enough hidden variables considered

**Cons**
- `hard`: non-convex optimization problems in general
Pros and Cons of Extraction Models

### Pros
- **‘easy’**: reduces human burden in designing features
- **powerful**: if enough hidden variables considered

### Cons
- **‘hard’**: non-convex optimization problems in general
- **overfitting**: needs proper regularization/validation
Pros and Cons of Extraction Models

Neural Network/Deep Learning

Pros
- ‘easy’: reduces human burden in designing features
- powerful: if enough hidden variables considered

Cons
- ‘hard’: non-convex optimization problems in general
- overfitting: needs proper regularization/validation

be careful when applying extraction models
Which of the following extraction model extracts Gaussian centers by \textit{k}-means and aggregate the Gaussians linearly?

1. RBF Network
2. Deep Learning
3. Adaptive Boosting
4. Matrix Factorization

\textbf{Reference Answer:} 1
Fun Time

Which of the following extraction model extracts Gaussian centers by \textit{k-means} and aggregate the Gaussians linearly?

1. RBF Network
2. Deep Learning
3. Adaptive Boosting
4. Matrix Factorization

Reference Answer: 1

Congratulations on being an expert in extraction models! :-)

Hsuan-Tien Lin (NTU CSIE)
Summary

1. Embedding Numerous Features: Kernel Models
2. Combining Predictive Features: Aggregation Models
3. Distilling Implicit Features: Extraction Models

Lecture 15: Matrix Factorization

- Linear Network Hypothesis
- feature extraction from binary vector encoding
- Basic Matrix Factorization
- alternating least squares between user/movie
- Stochastic Gradient Descent
- efficient and easily modified for practical use
- Summary of Extraction Models
  - powerful thus need careful use

• next: closing remarks of techniques