# Algorithm Foundations of Data Science and Engineering

## Lecture 9: Matrix Factorization

YANHAO WANG

DaSE @ ECNU
(for course related communications)
yhwang@dase.ecnu.edu.cn

Nov. 8, 2021

# Outline

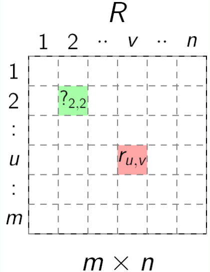
# Motivation

# Motivation

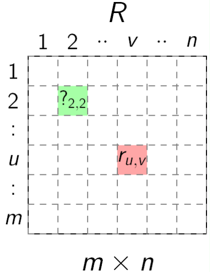
For recommender systems: a group of users give ratings to some items.

| User | Item | Rating |
|---|---|---|
| 1 | 5 | 100 |
| 1 | 10 | 80 |
| 1 | 13 | 30 |
| 2 | 10 | 50 |
| ... | ... | ... |
| $u$ | $v$ | $r$ |
| ... | ... | ... |

# Matrix with missing value

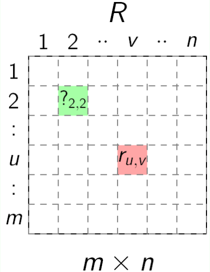


## Matrix with missing value



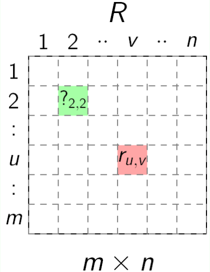

- $m, n$: numbers of users and items

# Matrix with missing value



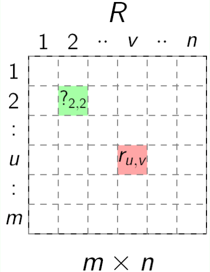

- $m, n$: numbers of users and items
- $u, v$: index for $u_{th}$ user and $v_{th}$ item

## Matrix with missing value




- $m, n$: 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

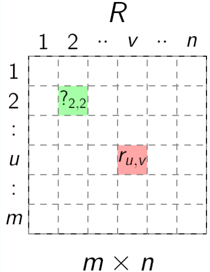
# Matrix with missing value




There are many missing values in the matrix, and many applications that can be modeled as the given matrix

- $m, n$: 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 with missing value

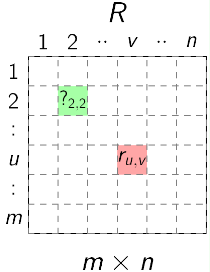



There are many missing values in the matrix, and many applications that can be modeled as the given matrix

- Product adoption

- $m, n$: 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 with missing value

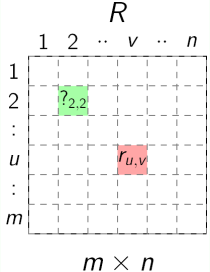



- $m, n$: numbers of users and items
- $u, v$: index for $u_{th}$ user and $v_{th}$ item
- $r_{u,v}$: $u_{th}$ user gives a rating $r_{u,v}$ to $v_{th}$ item

There are many missing values in the matrix, and many applications that can be modeled as the given matrix

- Product adoption
- APP download

# Matrix with missing value

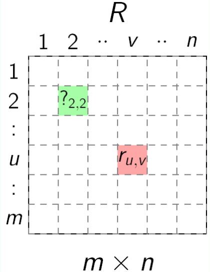



- $m, n$: numbers of users and items
- $u, v$: index for $u_{th}$ user and $v_{th}$ item
- $r_{u,v}$: $u_{th}$ user gives a rating $r_{u,v}$ to $v_{th}$ item

There are many missing values in the matrix, and many applications that can be modeled as the given matrix

- Product adoption
- APP download
- Music listening

# Matrix with missing value

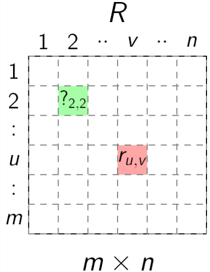



- $m, n$: numbers of users and items
- $u, v$: index for $u_{th}$ user and $v_{th}$ item
- $r_{u,v}$: $u_{th}$ user gives a rating $r_{u,v}$ to $v_{th}$ item

There are many missing values in the matrix, and many applications that can be modeled as the given matrix

- Product adoption
- APP download
- Music listening
- POI visiting

# Matrix with missing value

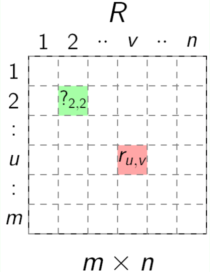



- $m, n$: numbers of users and items
- $u, v$: index for $u_{th}$ user and $v_{th}$ item
- $r_{u,v}$: $u_{th}$ user gives a rating $r_{u,v}$ to $v_{th}$ item

There are many missing values in the matrix, and many applications that can be modeled as the given matrix

- Product adoption
- APP download
- Music listening
- POI visiting
- Question addressing

# Matrix with missing value

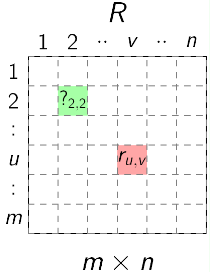



- $m, n$: numbers of users and items
- $u, v$: index for $u_{th}$ user and $v_{th}$ item
- $r_{u,v}$: $u_{th}$ user gives a rating $r_{u,v}$ to $v_{th}$ item

There are many missing values in the matrix, and many applications that can be modeled as the given matrix

- Product adoption
- APP download
- Music listening
- POI visiting
- Question addressing
- ...

# Matrix with missing value



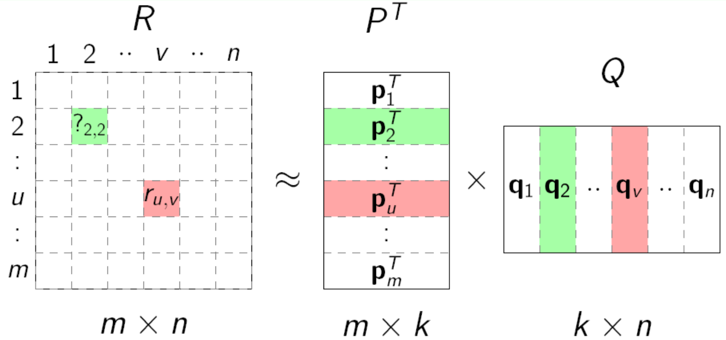

- $m, n$: numbers of users and items
- $u, v$: index for $u_{th}$ user and $v_{th}$ item
- $r_{u,v}$: $u_{th}$ user gives a rating $r_{u,v}$ to $v_{th}$ item

There are many missing values in the matrix, and many applications that can be modeled as the given matrix

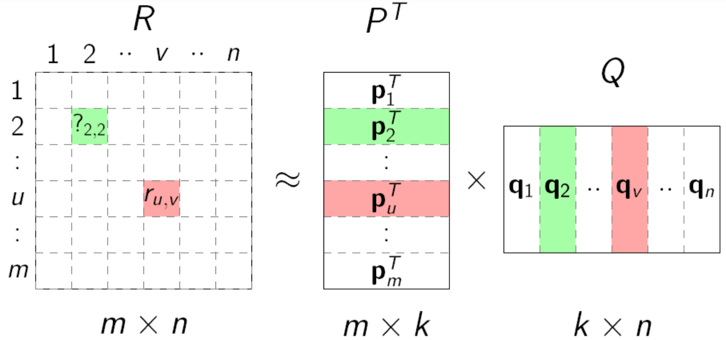
- Product adoption
- APP download
- Music listening
- POI visiting
- Question addressing
- ...

The given matrix can be used to model the online user behaviors.

# Predicting the missing values

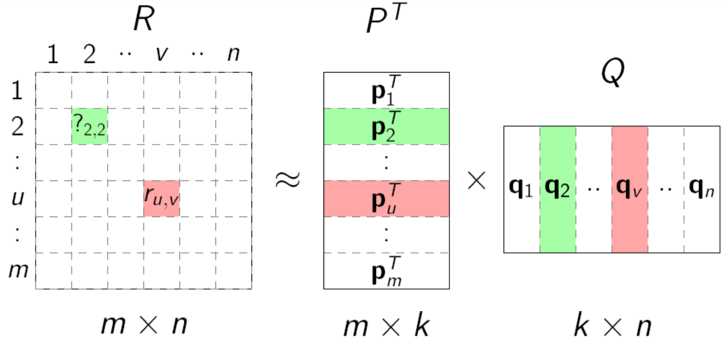


# Predicting the missing values



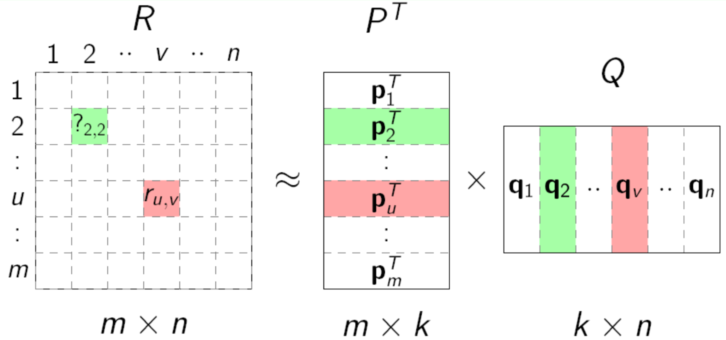

- $k$: number of latent dimensions.

# Predicting the missing values



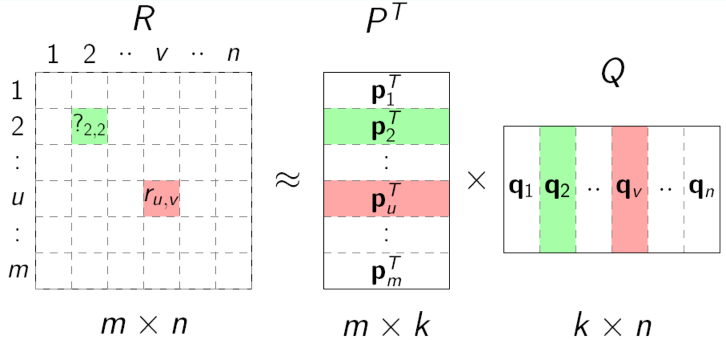

- $k$: number of latent dimensions.
- $r_{u,v} = \mathbf{p}_u^T \mathbf{q}_v$.

# Predicting the missing values




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

# Predicting the missing values




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

The approach is called matrix factorization, i.e., $R_{n \times m} \approx P_{n \times k} \cdot Q_{k \times m}$

# Optimization

Much of supervised machine learning can be written as an optimization problem

$$\min_{\mathbf{x}} \sum_{i=1}^{N} f(\mathbf{x}; y_i)$$

where $f$ is a loss function, $y_i$ is the training examples.

# Optimization

Much of supervised machine learning can be written as an optimization problem

$$\min_{\mathbf{x}} \sum_{i=1}^{N} f(\mathbf{x}; y_i)$$

where $f$ is a loss function, $y_i$ is the training examples.

- Example loss functions: logistic regression, linear regression, principle component analysis, neural network loss;

# Optimization

Much of supervised machine learning can be written as an optimization problem

$$\min_{\mathbf{x}} \sum_{i=1}^{N} f(\mathbf{x}; y_i)$$

where $f$ is a loss function, $y_i$ is the training examples.

- Example loss functions: logistic regression, linear regression, principle component analysis, neural network loss;
- Types of Optimization
  - Convex optimization: the easy case;

# Optimization

Much of supervised machine learning can be written as an optimization problem

$$\min_{\mathbf{x}} \sum_{i=1}^{N} f(\mathbf{x}; y_i)$$

where $f$ is a loss function, $y_i$ is the training examples.

- Example loss functions: logistic regression, linear regression, principle component analysis, neural network loss;
- Types of Optimization
  - Convex optimization: the easy case;
  - Non-convex optimization: NP-hard in general, includes deep learning.

# Outline

# Convex functions

**Definition**

Function $f(x)$ is a convex function if $\forall \alpha \in [0, 1]$,

$$f(\alpha x + (1 - \alpha) y) \leq \alpha f(x) + (1 - \alpha) f(y).$$

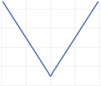
# Convex functions

## Definition

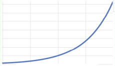
Function $f(x)$ is a convex function if $\forall\alpha \in [0,1]$,

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y).$$

$f(x) = x^2$



$f(x) = |x|$



$f(x) = e^x$

Examples of convex functions

- $$f(x) = x^2;$$

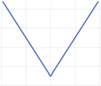
# Convex functions

## Definition

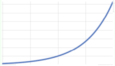
Function $f(x)$ is a convex function if $\forall \alpha \in [0,1]$,

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y).$$

$f(x) = x^2$



$f(x) = |x|$



$f(x) = e^x$

Examples of convex functions

- $$f(x) = x^2;$$
- $$f(x) = |x|;$$

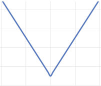
# Convex functions

## Definition

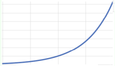
Function $f(x)$ is a convex function if $\forall\alpha \in [0,1]$,

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y).$$

$f(x) = x^2$



$f(x) = |x|$



$f(x) = e^x$

Examples of convex functions

- $$f(x) = x^2;$$
- $$f(x) = |x|;$$
- $$f(x) = e^x;$$

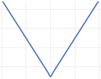
# Convex functions

## Definition

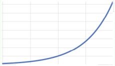
Function $f(x)$ is a convex function if $\forall \alpha \in [0, 1]$,

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y).$$

$f(x) = x^2$



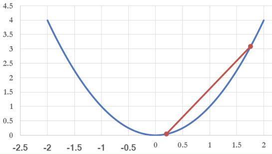
$f(x) = |x|$



$f(x) = e^x$

Examples of convex functions

- $$f(x) = x^2;$$
- $$f(x) = |x|;$$
- $$f(x) = e^x;$$



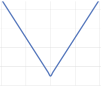
# Convex functions

## Definition

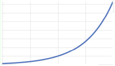
Function $f(x)$ is a convex function if $\forall \alpha \in [0, 1]$,

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y).$$

$f(x) = x^2$



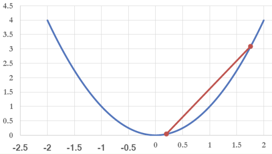
$f(x) = |x|$



$f(x) = e^x$

Examples of convex functions

- $$f(x) = x^2;$$
- $$f(x) = |x|;$$
- $$f(x) = e^x;$$




Any line segment we draw between two points lies above the curve.

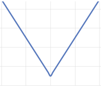
# Convex functions

## Definition

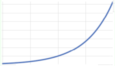
Function $f(x)$ is a convex function if $\forall \alpha \in [0,1]$,

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y).$$

$f(x) = x^2$



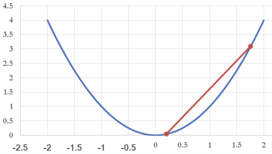
$f(x) = |x|$



$f(x) = e^x$

Examples of convex functions

- $$f(x) = x^2;$$
- $$f(x) = |x|;$$
- $$f(x) = e^x;$$




Any line segment we draw between two points lies above the curve.
Every local minimum is a global minimum.

## Properties of convex functions

- Non-negative combinations of convex functions are convex

$$h(x) = a \cdot f(x) + b \cdot g(x);$$

## Properties of convex functions

- Non-negative combinations of convex functions are convex

$$h(x) = a \cdot f(x) + b \cdot g(x);$$

- Affine scalings of convex functions are convex

$$h(x) = f(Ax + b);$$

# Properties of convex functions

- Non-negative combinations of convex functions are convex

$$h(x) = a \cdot f(x) + b \cdot g(x);$$

- Affine scalings of convex functions are convex

$$h(x) = f(Ax + b);$$

- Compositions of convex functions are **NOT** generally convex, except for convex nondecreasing functions.

$$h(x) = f(g(x)) \text{ (Neural nets are like this)};$$

For example, if $f(x) = e^{-x}$ on $[0, \infty)$, then $f$ is convex, but $f \circ f$ is concave.

# Outline

# Gradient descent

## Problem formulation

Consider unconstrained, smooth convex optimization

$$\min_{\mathbf{x}} f(\mathbf{x})$$

i.e., $f$ is convex and differentiable with $dom(f) = \mathbb{R}^n$.

# Gradient descent

## Problem formulation

Consider unconstrained, smooth convex optimization

$$\min_{\mathbf{x}} f(\mathbf{x})$$

i.e., $f$ is convex and differentiable with $dom(f) = \mathbb{R}^n$. Denote the optimal criterion value by $f^* = \min_{\mathbf{x}} f(\mathbf{x})$, and a solution by $\mathbf{x}^*$.

# Gradient descent

## Problem formulation

Consider unconstrained, smooth convex optimization

$$\min_{\mathbf{x}} f(\mathbf{x})$$

i.e., $f$ is convex and differentiable with $dom(f) = \mathbb{R}^n$. Denote the optimal criterion value by $f^* = \min_{\mathbf{x}} f(\mathbf{x})$, and a solution by $\mathbf{x}^*$.

## Gradient descent algorithm

# Gradient descent

## Problem formulation

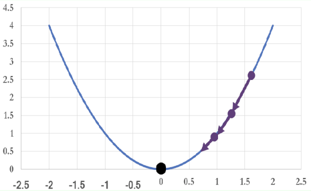
Consider unconstrained, smooth convex optimization

$$\min_{\mathbf{x}} f(\mathbf{x})$$

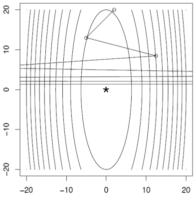
i.e., $f$ is convex and differentiable with $dom(f) = \mathbb{R}^n$. Denote the optimal criterion value by $f^* = \min_{\mathbf{x}} f(\mathbf{x})$, and a solution by $\mathbf{x}^*$.

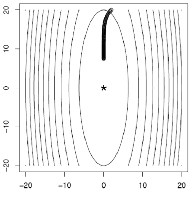
## Gradient descent algorithm

1: Pick a starting point $\mathbf{x}^{(0)} \in \mathbb{R}^n$;
2: For $k = 1, 2, \cdots$;
3: $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - \lambda \cdot \nabla f(\mathbf{x}^{(k-1)})$;
where $\lambda$ is the step size.

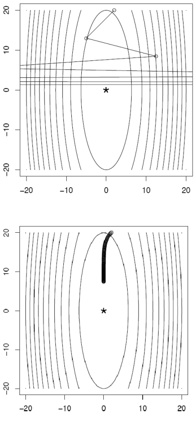


# How to choose step size



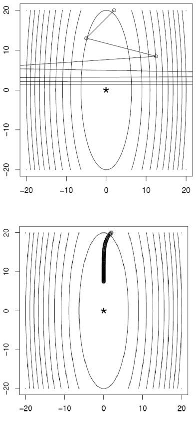



# How to choose step size




Consider function $f(\mathbf{x}) = (10x_1^2 + x_2^2)/2$,

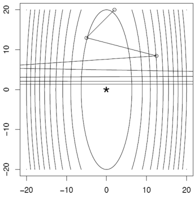
# How to choose step size

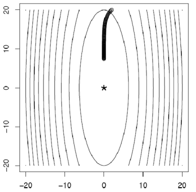



Consider function $f(\mathbf{x}) = (10x_1^2 + x_2^2)/2$,

- Simply take a large value of $\lambda$, it will slowly converge (after 8 iterations).

# How to choose step size

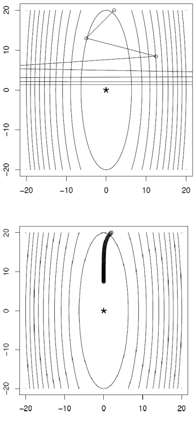



Consider function $f(\mathbf{x}) = (10x_1^2 + x_2^2)/2$,

- Simply take a large value of $\lambda$, it will slowly converge (after 8 iterations).
- It can also be slow if $\lambda$ is to small (after 100 iterations).

## How to choose step size




(Refer to [a])

Consider function $f(\mathbf{x}) = (10x_1^2 + x_2^2)/2$,

- Simply take a large value of $\lambda$, it will slowly converge (after 8 iterations).
- It can also be slow if $\lambda$ is to small (after 100 iterations).
- When $f$ is additionally Lipschitz continuous with constant $L > 0$, the gradient descent has convergence rate $O(1/k)$, i.e., to get $f(\mathbf{x}^{(k)}) - f^* \leq \epsilon$, we need $O(\frac{1}{\epsilon})$ iterations.

[a] https://www.cs.rochester.edu/u/jliu/CSC-576/class-note-10.pdf

## The problem with Gradient descent

In many machine learning task, the cost function can be written as

$$h(\mathbf{x}) = \sum_{i=1}^{N} f(\mathbf{x}; y_i),$$

where $f$ is the loss function, $y_i$ is a training example.

## The problem with Gradient descent

In many machine learning task, the cost function can be written as

$$h(\mathbf{x}) = \sum_{i=1}^{N} f(\mathbf{x}; y_i),$$

where $f$ is the loss function, $y_i$ is a training example.
For large-scale optimization, computing the gradient takes $O(N)$ time

$$\bigtriangledown h(x) = \frac{1}{N} \sum_{i=1}^{N} \bigtriangledown f(\mathbf{x}; y_i).$$

# The problem with Gradient descent

In many machine learning task, the cost function can be written as

$$h(\mathbf{x}) = \sum_{i=1}^{N} f(\mathbf{x}; y_i),$$

where $f$ is the loss function, $y_i$ is a training example.
For large-scale optimization, computing the gradient takes $O(N)$ time

$$\triangledown h(x) = \frac{1}{N} \sum_{i=1}^{N} \triangledown f(\mathbf{x}; y_i).$$

If we want to scale up to huge datasets, so how can we do this? (Refer to [a])

[a] http://www.cs.cornell.edu/courses/cs6787/2017fa/Lecture1.pdf

## Stochastic Gradient Descent

Idea: rather than using the full gradient, just use one training example.

# Stochastic Gradient Descent

Idea: rather than using the full gradient, just use one training example.

- Super fast to compute

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \lambda \bigtriangledown f(\mathbf{x}; y_{\tilde{i}_t}),$$

  where $y_{\tilde{i}_t}$ is an example selected uniformly at random from the data set.

## Stochastic Gradient Descent

Idea: rather than using the full gradient, just use one training example.

- Super fast to compute

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \lambda \bigtriangledown f(\mathbf{x}; y_{\tilde{i}_t}),$$

  where $y_{\tilde{i}_t}$ is an example selected uniformly at random from the data set.

- In expectation, it is just gradient descent

$$E(\mathbf{x}_{t+1}) = E(\mathbf{x}_t) - \lambda E(\bigtriangledown f(\mathbf{x}; y_{\tilde{i}_t})) = E(\mathbf{x}_t) - \alpha \frac{1}{N} \sum_{i=1}^{N} \bigtriangledown f(\mathbf{x}; y_i)$$

## Stochastic Gradient Descent

Idea: rather than using the full gradient, just use one training example.

- Super fast to compute

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \lambda \bigtriangledown f(\mathbf{x}; y_{\tilde{i}_t}),$$

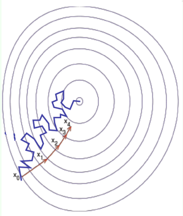
  where $y_{\tilde{i}_t}$ is an example selected uniformly at random from the data set.

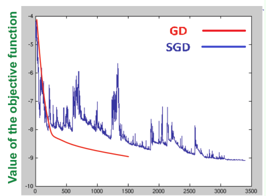
- In expectation, it is just gradient descent

$$E(\mathbf{x}_{t+1}) = E(\mathbf{x}_t) - \lambda E(\bigtriangledown f(\mathbf{x}; y_{\tilde{i}_t})) = E(\mathbf{x}_t) - \alpha \frac{1}{N} \sum_{i=1}^{N} \bigtriangledown f(\mathbf{x}; y_i)$$

- Can SGD converge using just one example to estimate the gradient?

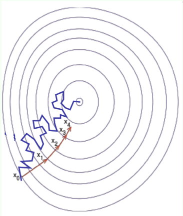
# SGD VS. GD

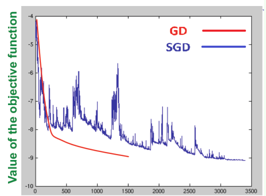



- GD improves the value of the objective function at every step.

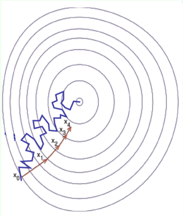
# SGD VS. GD

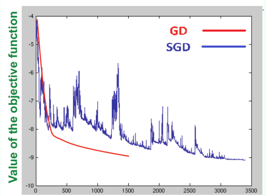





- GD improves the value of the objective function at every step.
- SGD improves the value but in a "noisy" way.

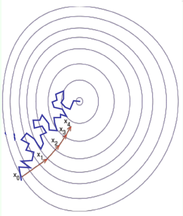
# SGD VS. GD

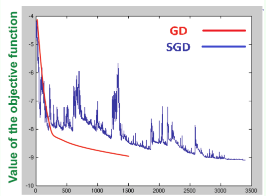



- GD improves the value of the objective function at every step.
- SGD improves the value but in a "noisy" way.
- GD takes fewer steps to converge but each step takes much longer to compute.

# SGD VS. GD




- GD improves the value of the objective function at every step.
- SGD improves the value but in a "noisy" way.
- GD takes fewer steps to converge but each step takes much longer to compute.
- In practice, SGD is much faster!

# Matrix factorization

## Motivation

How can one determine the factor matrices $P$ and $Q$, so that the fully specified matrix $R$ matches $PQ^T$ as closely as possible?

# Matrix factorization

## Motivation

How can one determine the factor matrices $P$ and $Q$, so that the fully specified matrix $R$ matches $PQ^T$ as closely as possible?

Formulate an optimization problem as:

$$\text{Minimize } J = \frac{1}{2}\|R - PQ^T\|_F^2$$

$$s.t. \text{ No constraints on } P \text{ and } Q$$

where $\|\cdot\|_F^2$ represents the squared Frobenius norm of the matrix.

# Matrix factorization

## Motivation

How can one determine the factor matrices $P$ and $Q$, so that the fully specified matrix $R$ matches $PQ^T$ as closely as possible?

Formulate an optimization problem as:

$$\text{Minimize } J = \frac{1}{2}\|R - PQ^T\|_F^2$$

$$s.t. \text{ No constraints on } P \text{ and } Q$$

where $\|\cdot\|_F^2$ represents the squared Frobenius norm of the matrix.

- Thus, the objective function is equal to the sum of the squares of the entries in the residual matrix $R - PQ^T$.

# Matrix factorization

## Motivation

How can one determine the factor matrices $P$ and $Q$, so that the fully specified matrix $R$ matches $PQ^T$ as closely as possible?

Formulate an optimization problem as:

$$\text{Minimize } J = \frac{1}{2}\|R - PQ^T\|_F^2$$

$$s.t. \text{ No constraints on } P \text{ and } Q$$

where $\|\cdot\|_F^2$ represents the squared Frobenius norm of the matrix.

- Thus, the objective function is equal to the sum of the squares of the entries in the residual matrix $R - PQ^T$.
- This objective function can be viewed as a quadratic loss function, which quantifies the loss of accuracy in estimating the matrix $R$ with the use of low-rank factorization.

## Matrix factorization

### Definition

Given a set of users $U$, and a set of items $D$, let $R \in \mathbb{R}^{|U| \times |D|}$ be the rating matrix.

# Matrix factorization

**Definition**

Given a set of users $U$, and a set of items $D$, let $R \in \mathbb{R}^{|U| \times |D|}$ be the rating matrix. The matrix factorization is to find two matrices $P \in \mathbb{R}^{|U| \times K}$ and $Q \in \mathbb{R}^{|D| \times K}$ such that $R \approx PQ^T = \widehat{R}$, where $K$ denotes the dimensionality of latent features.

# Matrix factorization

**Definition**

Given a set of users $U$, and a set of items $D$, let $R \in \mathbb{R}^{|U| \times |D|}$ be the rating matrix. The matrix factorization is to find two matrices $P \in \mathbb{R}^{|U| \times K}$ and $Q \in \mathbb{R}^{|D| \times K}$ such that $R \approx PQ^T = \widehat{R}$, where $K$ denotes the dimensionality of latent features.

- Each row of $P$ would represent the strength of the associations between a user and the features.

# Matrix factorization

**Definition**

Given a set of users $U$, and a set of items $D$, let $R \in \mathbb{R}^{|U|\times|D|}$ be the rating matrix. The matrix factorization is to find two matrices $P \in \mathbb{R}^{|U|\times K}$ and $Q \in \mathbb{R}^{|D|\times K}$ such that $R \approx PQ^T = \widehat{R}$, where $K$ denotes the dimensionality of latent features.

- Each row of $P$ would represent the strength of the associations between a user and the features.
- Each row of $Q$ would represent the strength of the associations between an item and the features.

# Matrix factorization

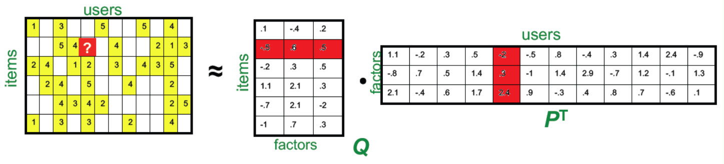
**Definition**

Given a set of users $U$, and a set of items $D$, let $R \in \mathbb{R}^{|U| \times |D|}$ be the rating matrix. The matrix factorization is to find two matrices $P \in \mathbb{R}^{|U| \times K}$ and $Q \in \mathbb{R}^{|D| \times K}$ such that $R \approx PQ^T = \widehat{R}$, where $K$ denotes the dimensionality of latent features.

- Each row of $P$ would represent the strength of the associations between a user and the features.
- Each row of $Q$ would represent the strength of the associations between an item and the features.
- Now, we have to find a way to obtain $P$ and $Q$.

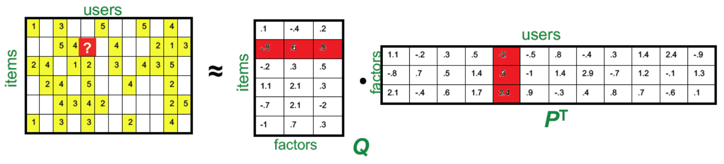
# Matrix factorization Cont'd

## Example



# Matrix factorization Cont'd

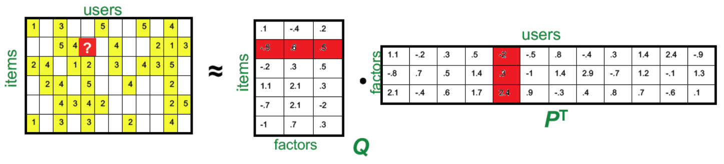
## Example




Using $P$ and $Q$, how to estimate the missing rating of user $u$ for item $i$?

# Matrix factorization Cont'd

## Example

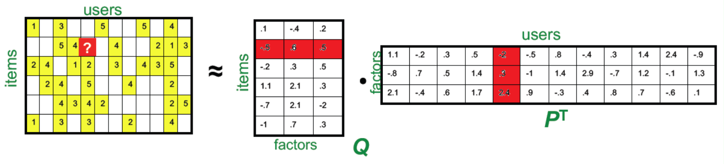



Using $P$ and $Q$, how to estimate the missing rating of user $u$ for item $i$?

$$\widehat{r}_{ui} = \mathbf{q}_i^T \mathbf{P}_u = \sum_{j=1}^{k} p_{uj} q_{ji}.$$

# Matrix factorization Cont'd

## Example




Using $P$ and $Q$, how to estimate the missing rating of user $u$ for item $i$?

$$\widehat{r}_{ui} = \mathbf{q}_i^T \mathbf{P}_u = \sum_{j=1}^{k} p_{uj} q_{ji}.$$

The estimating error can be

$$e_{ui} = r_{ui} - \widehat{r}_{ui} = r_{ui} - \sum_{j=1}^{k} p_{uj} q_{ji}.$$

## Problem formulation

### Formal definition

$$J = \min_{P,Q} \frac{1}{2} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mathbf{p}_u^T \mathbf{q}_i)^2,$$

where $r_{ui}$ is the known rating of user $u$ for item $i$, $\widehat{u}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$ is the predicted rating given by user $u$ for item $i$, and the set of all user-item pair $(u, i)$, which are observed in $R$, be denoted by $\mathcal{K}$, i.e., $\mathcal{K} = \{(u, i) | r_{ui} \text{ is observed}\}$.

# Problem formulation

## Formal definition

$$J = \min_{P,Q} \frac{1}{2} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mathbf{p}_u^T \mathbf{q}_i)^2,$$

where $r_{ui}$ is the known rating of user $u$ for item $i$, $\widehat{u}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$ is the predicted rating given by user $u$ for item $i$, and the set of all user-item pair $(u, i)$, which are observed in $R$, be denoted by $\mathcal{K}$, i.e., $\mathcal{K} = \{(u, i) | r_{ui} \text{ is observed}\}$.

- The error between the predicted rating and the real rating, can be calculated by the following equation for each user-item pair: $e_{ui} = r_{ui} - \widehat{r}_{ui} = r_{ui} - \sum_{j=1}^{k} p_{uj} q_{ji}$.

# Problem formulation

## Formal definition

$$J = \min_{P,Q} \frac{1}{2} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mathbf{p}_u^T \mathbf{q}_i)^2,$$

where $r_{ui}$ is the known rating of user $u$ for item $i$, $\widehat{u}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$ is the predicted rating given by user $u$ for item $i$, and the set of all user-item pair $(u, i)$, which are observed in $R$, be denoted by $\mathcal{K}$, i.e., $\mathcal{K} = \{(u, i) | r_{ui} \text{ is observed}\}$.

- The error between the predicted rating and the real rating, can be calculated by the following equation for each user-item pair: $e_{ui} = r_{ui} - \widehat{r}_{ui} = r_{ui} - \sum_{j=1}^{k} p_{uj} q_{ji}$.
- Now, we have to find a way to obtain $P$ and $Q$.

# Outline

# Batch gradient descent algorithm

## Estimate parameters iteratively

Let's start at $P^{(0)}$ and $Q^{(0)}$.

# Batch gradient descent algorithm

## Estimate parameters iteratively

Let's start at $P^{(0)}$ and $Q^{(0)}$.

- $\frac{\partial}{\partial p_{uj}} e_{ui}^2 = -(r_{ui} - \widehat{r}_{ui}) q_{ji}$ and $\frac{\partial}{\partial q_{ji}} e_{ui}^2 = -(r_{ui} - \widehat{r}_{ui}) p_{uj}$.

# Batch gradient descent algorithm

## Estimate parameters iteratively

Let's start at $P^{(0)}$ and $Q^{(0)}$.

- $\frac{\partial}{\partial p_{uj}} e_{ui}^2 = -(r_{ui} - \widehat{r}_{ui}) q_{ji}$ and $\frac{\partial}{\partial q_{ji}} e_{ui}^2 = -(r_{ui} - \widehat{r}_{ui}) p_{uj}$.
- Update rules:
  - $p_{uj}^{(t+1)} \leftarrow p_{uj}^{(t)} + \alpha \sum_{i:(u,i) \in \mathcal{K}} e_{ui}^{(t)} q_{ji}^{(t)}$.

# Batch gradient descent algorithm

## Estimate parameters iteratively

Let's start at $P^{(0)}$ and $Q^{(0)}$.

- $\frac{\partial}{\partial p_{uj}} e_{ui}^2 = -(r_{ui} - \widehat{r}_{ui}) q_{ji}$ and $\frac{\partial}{\partial q_{ji}} e_{ui}^2 = -(r_{ui} - \widehat{r}_{ui}) p_{uj}$.
- Update rules:
  - $p_{uj}^{(t+1)} \leftarrow p_{uj}^{(t)} + \alpha \sum_{i:(u,i)\in\mathcal{K}} e_{ui}^{(t)} q_{ji}^{(t)}$.
  - $q_{ji}^{(t+1)} \leftarrow q_{ji}^{(t)} + \alpha \sum_{u:(u,i)\in\mathcal{K}} e_{ui}^{(t)} p_{uj}^{(t)}$.
  - Where $e_{ui}^{(t)} = r_{ui} - {p_u^{(t)}}^T q_i^{(t)}$

# Batch gradient descent algorithm

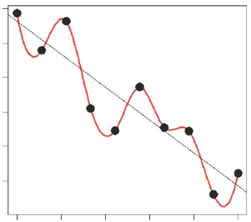
**Estimate parameters iteratively**

Let's start at $P^{(0)}$ and $Q^{(0)}$.

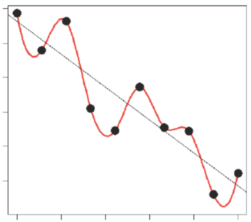
- $\frac{\partial}{\partial p_{uj}} e_{ui}^2 = -(r_{ui} - \widehat{r}_{ui}) q_{ji}$ and $\frac{\partial}{\partial q_{ji}} e_{ui}^2 = -(r_{ui} - \widehat{r}_{ui}) p_{uj}$.
- Update rules:
  - $p_{uj}^{(t+1)} \leftarrow p_{uj}^{(t)} + \alpha \sum_{i:(u,i)\in\mathcal{K}} e_{ui}^{(t)} q_{ji}^{(t)}$.
  - $q_{ji}^{(t+1)} \leftarrow q_{ji}^{(t)} + \alpha \sum_{u:(u,i)\in\mathcal{K}} e_{ui}^{(t)} p_{uj}^{(t)}$.
  - Where $e_{ui}^{(t)} = r_{ui} - {p_u^{(t)}}^T q_i^{(t)}$
- Subsequently, the updates can be computed as follows:
  - $P^{(t+1)} \leftarrow P^{(t)} + \alpha E^{(t)} Q^{(t)}$.
  - $Q^{(t+1)} \leftarrow Q^{(t)} + \alpha {E^{(t)}}^T P^{(t)}$.

# Outline

# Matrix factorization: regularization

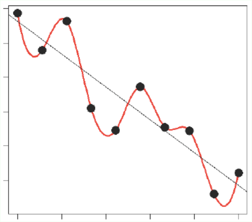


# Matrix factorization: regularization



- The observed set $\mathcal{K}$ of ratings is small, which can cause overfitting (also common in classification problem).

# Matrix factorization: regularization



- The observed set $\mathcal{K}$ of ratings is small, which can cause overfitting (also common in classification problem).
- Regularization is a common approach to address the problem.

## Regularization

$$\min_{q^*,p^*} J = \frac{1}{2}\Big[\sum_{(u,i)\in\mathcal{K}} (r_{ui} - q_i^T p_u)^2 + \lambda(\|Q\|^2 + \|P\|^2)\Big],$$

# Gradient descent algorithm for regularization

## Batch gradient descent

- $p_{uj}^{(t+1)} \leftarrow p_{uj}^{(t)} + \alpha(\sum_{i:(u,i)\in\mathcal{K}} e_{ui}^{(t)} q_{ji}^{(t)} - \lambda p_{uj}^{(t)})$.

# Gradient descent algorithm for regularization

## Batch gradient descent

- $p_{uj}^{(t+1)} \leftarrow p_{uj}^{(t)} + \alpha(\sum_{i:(u,i)\in\mathcal{K}} e_{ui}^{(t)} q_{ji}^{(t)} - \lambda p_{uj}^{(t)})$.
- $q_{ji}^{(t+1)} \leftarrow q_{ji}^{(t)} + \alpha(\sum_{u:(u,i)\in\mathcal{K}} e_{ui}^{(t)} p_{uj}^{(t)} - \lambda q_{ji}^{(t)})$,

  where $e_{ui}^{(t)} = r_{ui} - \mathbf{p}_u^{(t)\,T} \mathbf{q}_i^{(t)}$

# Gradient descent algorithm for regularization

## Batch gradient descent

- $p_{uj}^{(t+1)} \leftarrow p_{uj}^{(t)} + \alpha(\sum_{i:(u,i)\in\mathcal{K}} e_{ui}^{(t)} q_{ji}^{(t)} - \lambda p_{uj}^{(t)})$.
- $q_{ji}^{(t+1)} \leftarrow q_{ji}^{(t)} + \alpha(\sum_{u:(u,i)\in\mathcal{K}} e_{ui}^{(t)} p_{uj}^{(t)} - \lambda q_{ji}^{(t)})$,

  where $e_{ui}^{(t)} = r_{ui} - \mathbf{p}_u^{(t)T} \mathbf{q}_i^{(t)}$

## Stochastic gradient descent

- $p_{uj}^{(t+1)} \leftarrow p_{uj}^{(t)} + \alpha(e_{ui}^{(t)} q_{ji}^{(t)} - \lambda p_{uj}^{(t)})$.

# Gradient descent algorithm for regularization

## Batch gradient descent

- $p_{uj}^{(t+1)} \leftarrow p_{uj}^{(t)} + \alpha(\sum_{i:(u,i)\in\mathcal{K}} e_{ui}^{(t)} q_{ji}^{(t)} - \lambda p_{uj}^{(t)})$.
- $q_{ji}^{(t+1)} \leftarrow q_{ji}^{(t)} + \alpha(\sum_{u:(u,i)\in\mathcal{K}} e_{ui}^{(t)} p_{uj}^{(t)} - \lambda q_{ji}^{(t)})$,

  where $e_{ui}^{(t)} = r_{ui} - \mathbf{p}_u^{(t)^T} \mathbf{q}_i^{(t)}$

## Stochastic gradient descent

- $p_{uj}^{(t+1)} \leftarrow p_{uj}^{(t)} + \alpha(e_{ui}^{(t)} q_{ji}^{(t)} - \lambda p_{uj}^{(t)})$.
- $q_{ji}^{(t+1)} \leftarrow q_{ji}^{(t)} + \alpha(e_{ui}^{(t)} p_{uj}^{(t)} - \lambda q_{ji}^{(t)})$,

  where $e_{ui}^{(t)} = r_{ui} - \mathbf{p}_u^{(t)^T} \mathbf{q}_i^{(t)}$

# Incorporating user and item biases

## Loss function

$$J = \frac{1}{2}\Big[\sum_{(u,i)\in\mathcal{K}} (r_{ui} - b_i - b_u - q_i^T p_u)^2 + \lambda(\|Q\|^2 + \|P\|^2 + \|b_u\|^2 + \|b_i\|^2)\Big]$$

# Incorporating user and item biases

## Loss function

$$J = \frac{1}{2}\Big[\sum_{(u,i)\in\mathcal{K}} (r_{ui} - b_i - b_u - q_i^T p_u)^2 + \lambda(\|Q\|^2 + \|P\|^2 + \|b_u\|^2 + \|b_i\|^2)\Big]$$

- Instead of having separate bias variables $b_u$ and $b_i$ for users and items, we can increase the size of the factor matrices to incorporate these bias variables.

# Incorporating user and item biases

## Loss function

$$J = \frac{1}{2}\Big[\sum_{(u,i)\in\mathcal{K}} (r_{ui} - b_i - b_u - q_i^T p_u)^2 + \lambda(\|Q\|^2 + \|P\|^2 + \|b_u\|^2 + \|b_i\|^2)\Big]$$

- Instead of having separate bias variables $b_u$ and $b_i$ for users and items, we can increase the size of the factor matrices to incorporate these bias variables.
  - $p_{u(k+1)} = b_u$ and $p_{u(k+2)} = 1, \forall u \in \{1, 2, \cdots, n\}$
  - $q_{i(k+1)} = 1$ and $p_{i(k+2)} = b_i, \forall i \in \{1, 2, \cdots, m\}$

# Incorporating user and item biases

## Loss function

$$J = \frac{1}{2}\Big[\sum_{(u,i)\in\mathcal{K}} (r_{ui} - b_i - b_u - q_i^T p_u)^2 + \lambda(\|Q\|^2 + \|P\|^2 + \|b_u\|^2 + \|b_i\|^2)\Big]$$

- Instead of having separate bias variables $b_u$ and $b_i$ for users and items, we can increase the size of the factor matrices to incorporate these bias variables.
  - $p_{u(k+1)} = b_u$ and $p_{u(k+2)} = 1, \forall u \in \{1, 2, \cdots, n\}$
  - $q_{i(k+1)} = 1$ and $p_{i(k+2)} = b_i, \forall i \in \{1, 2, \cdots, m\}$
-

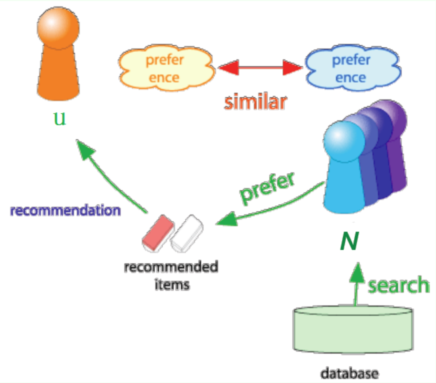
$$\min_{q^*, p^*} J = \frac{1}{2}\Big[\sum_{(u,i)\in\mathcal{K}} (r_{ui} - \widetilde{q}_i^T \widetilde{p}_u)^2 + \lambda(\|\widetilde{Q}\|^2 + \|\widetilde{P}\|^2)\Big]$$

$$s.t. (k+2)\text{th column of } \widetilde{P} \text{ contains only 1s}$$

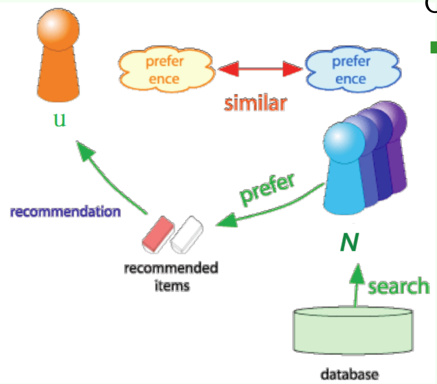
$$(k+1)\text{th column of } \widetilde{Q} \text{ contains only 1s}$$

# Outline

# Motivation of collaborative filtering



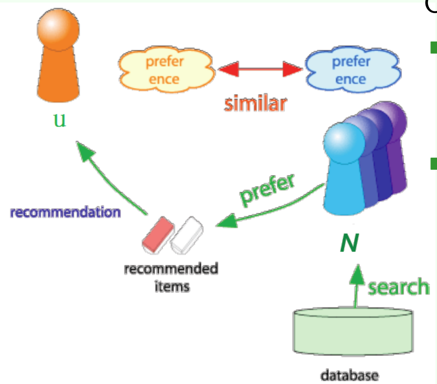
# Motivation of collaborative filtering




Consider user $U$

- Find set $S$ of other users whose ratings are "similar" to $u'$s ratings.

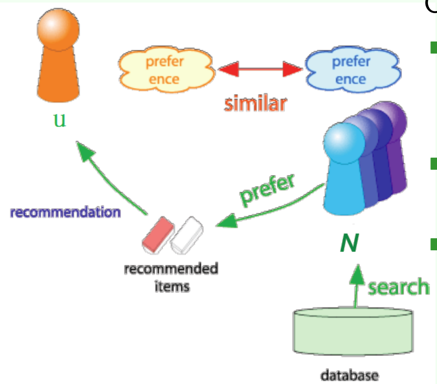
# Motivation of collaborative filtering




Consider user $U$

- Find set $S$ of other users whose ratings are "similar" to $u'$s ratings.
- Estimate $u'$s ratings based on ratings of users in $S$.

# Motivation of collaborative filtering




Consider user $U$

- Find set $S$ of other users whose ratings are "similar" to $u'$s ratings.
- Estimate $u'$s ratings based on ratings of users in $S$.
- Question: how to evaluate whose ratings are "similar" to $u'$s ratings.

# MF for collaborative filtering

## Problem formulation

$$J = \min_{P,Q} \frac{1}{2} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mathbf{p}_u^T \mathbf{q}_i)^2 + \frac{\gamma}{2} \sum_{(u,v)\in\mathcal{U}} s_{uv} \|\mathbf{p}_u - \mathbf{p}_v\|_2^2,$$

where $r_{ui}$ is the known rating of user $u$ for item $i$, $\widehat{u}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$ is the predicted rating given by user $u$ for item $i$, and $s_{uv}$ is the preference similarity between users $u$ and $v$.

# MF for collaborative filtering

## Problem formulation

$$J = \min_{P,Q} \frac{1}{2} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mathbf{p}_u^T \mathbf{q}_i)^2 + \frac{\gamma}{2} \sum_{(u,v)\in\mathcal{U}} s_{uv} \|\mathbf{p}_u - \mathbf{p}_v\|_2^2,$$

where $r_{ui}$ is the known rating of user $u$ for item $i$, $\widehat{u}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$ is the predicted rating given by user $u$ for item $i$, and $s_{uv}$ is the preference similarity between users $u$ and $v$.

### Stochastic gradient descent

- $p_{uj}^{(t+1)} \leftarrow p_{uj}^{(t)} + \alpha\big(e_{ui}^{(t)} q_{ji}^{(t)} - \gamma s_{uv} (p_{uj}^{(t)} - p_{vj}^{(t)}) p_{uj}^{(t)}\big).$

# MF for collaborative filtering

## Problem formulation

$$J = \min_{P,Q} \frac{1}{2} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mathbf{p}_u^T \mathbf{q}_i)^2 + \frac{\gamma}{2} \sum_{(u,v)\in\mathcal{U}} s_{uv} \|\mathbf{p}_u - \mathbf{p}_v\|_2^2,$$

where $r_{ui}$ is the known rating of user $u$ for item $i$, $\widehat{u}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$ is the predicted rating given by user $u$ for item $i$, and $s_{uv}$ is the preference similarity between users $u$ and $v$.

### Stochastic gradient descent

- $p_{uj}^{(t+1)} \leftarrow p_{uj}^{(t)} + \alpha\big(e_{ui}^{(t)} q_{ji}^{(t)} - \gamma s_{uv}(p_{uj}^{(t)} - p_{vj}^{(t)}) p_{uj}^{(t)}\big).$
- $q_{ji}^{(t+1)} \leftarrow q_{ji}^{(t)} + \alpha e_{ui}^{(t)} p_{uj}^{(t)},$

  where $e_{ui}^{(t)} = r_{ui} - \mathbf{p}_u^{(t)\,T} \mathbf{q}_i^{(t)}$

# Non-negative matrix factorization

## Formulation

Define an optimization problem as:

$$\begin{aligned}\text{Minimize } J &= \frac{1}{2}\|R - UV^T\|_F^2 \\ s.t. U &\geq 0 \\ V &\geq 0\end{aligned}$$

# Non-negative matrix factorization

## Formulation

Define an optimization problem as:

$$\text{Minimize } J = \frac{1}{2}\|R - UV^T\|_F^2$$
$$s.t. U \geq 0$$
$$V \geq 0$$

- Users specify a "like" for an item, but no mechanism to specify a "dislike", such as browsing or buy behaviors, Web-pages clicking, and Facebook liking, etc.

# Non-negative matrix factorization

## Formulation

Define an optimization problem as:

$$\begin{aligned} \text{Minimize } J = \frac{1}{2}\|R - UV^T\|_F^2 \\ s.t. U \geq 0 \\ V \geq 0 \end{aligned}$$

- Users specify a “like” for an item, but no mechanism to specify a “dislike”, such as browsing or buy behaviors, Web-pages clicking, and Facebook liking, etc.
- Many attributes of entities are non-negative.
  - Pixels of an image

# Non-negative matrix factorization

## Formulation

Define an optimization problem as:

$$\begin{aligned} \text{Minimize } J = \frac{1}{2}\|R - UV^T\|_F^2 \\ s.t. U \geq 0 \\ V \geq 0 \end{aligned}$$

- Users specify a "like" for an item, but no mechanism to specify a "dislike", such as browsing or buy behaviors, Web-pages clicking, and Facebook liking, etc.
- Many attributes of entities are non-negative.
  - Pixels of an image
  - Frequencies of words in a document

# Non-negative matrix factorization

## Formulation

Define an optimization problem as:

$$\text{Minimize } J = \frac{1}{2}\|R - UV^T\|_F^2$$
$$s.t. U \geq 0$$
$$V \geq 0$$

- Users specify a "like" for an item, but no mechanism to specify a "dislike", such as browsing or buy behaviors, Web-pages clicking, and Facebook liking, etc.
- Many attributes of entities are non-negative.
  - Pixels of an image
  - Frequencies of words in a document
  - Prices of stocks

# Solution for NMF

## Iterative algorithm

- $u_{ij}^{(t+1)} \leftarrow \frac{(RV^{(t)})_{ij} u_{ij}^{(t)}}{(U^{(t)} V^{(t)T} V^{(t)})_{ij} + \epsilon}$, $v_{ij}^{(t+1)} \leftarrow \frac{(R^T U^{(t)})_{ij} v_{ij}^{(t)}}{(V^{(t)} U^{(t)T} U^{(t)})_{ij} + \epsilon}$, where $\epsilon$ is a small term, e.g., $10^{-9}$.

# Solution for NMF

## Iterative algorithm

- $u_{ij}^{(t+1)} \leftarrow \frac{(RV^{(t)})_{ij} u_{ij}^{(t)}}{(U^{(t)} V^{(t)T} V^{(t)})_{ij} + \epsilon}$, $v_{ij}^{(t+1)} \leftarrow \frac{(R^T U^{(t)})_{ij} v_{ij}^{(t)}}{(V^{(t)} U^{(t)T} U^{(t)})_{ij} + \epsilon}$, where $\epsilon$ is a small term, e.g., $10^{-9}$.

## Regularization

As in the case of other types of matrix factorization, regularization can be used to improve the quality of the underlying solution.

# Solution for NMF

## Iterative algorithm

- $u_{ij}^{(t+1)} \leftarrow \frac{(RV^{(t)})_{ij} u_{ij}^{(t)}}{(U^{(t)} V^{(t)T} V^{(t)})_{ij} + \epsilon}$, $v_{ij}^{(t+1)} \leftarrow \frac{(R^T U^{(t)})_{ij} v_{ij}^{(t)}}{(V^{(t)} U^{(t)T} U^{(t)})_{ij} + \epsilon}$, where $\epsilon$ is a small term, e.g., $10^{-9}$.

## Regularization

As in the case of other types of matrix factorization, regularization can be used to improve the quality of the underlying solution.

- The basic idea is to add the penalties $\frac{\lambda_1 \|U\|^2}{2} + \frac{\lambda_2 \|V\|^2}{2}$ to the objective function.

# Solution for NMF

## Iterative algorithm

- $u_{ij}^{(t+1)} \leftarrow \frac{(RV^{(t)})_{ij} u_{ij}^{(t)}}{(U^{(t)} V^{(t)T} V^{(t)})_{ij} + \epsilon}$, $v_{ij}^{(t+1)} \leftarrow \frac{(R^T U^{(t)})_{ij} v_{ij}^{(t)}}{(V^{(t)} U^{(t)T} U^{(t)})_{ij} + \epsilon}$, where $\epsilon$ is a small term, e.g., $10^{-9}$.

## Regularization

As in the case of other types of matrix factorization, regularization can be used to improve the quality of the underlying solution.

- The basic idea is to add the penalties $\frac{\lambda_1 \|U\|^2}{2} + \frac{\lambda_2 \|V\|^2}{2}$ to the objective function.
- The update rules:
  - $u_{ij}^{(t+1)} \leftarrow \max\left\{ \left\lceil \frac{(RV^{(t)})_{ij} - \lambda_1 u_{ij}^{(t)}}{(U^{(t)} V^{(t)T} V^{(t)})_{ij} + \epsilon} \right\rceil u_{ij}^{(t)}, 0 \right\}$
  - $v_{ij}^{(t+1)} \leftarrow \max\left\{ \left\lceil \frac{(R^T U^{(t)})_{ij} - \lambda_2 v_{ij}^{(t)}}{(V^{(t)} U^{(t)T} U^{(t)})_{ij} + \epsilon} \right\rceil v_{ij}^{(t)}, 0 \right\}$

# Take-home messages

- Motivation
- Gradient Descent
  - Convex Functions
  - Gradient Descent
- Matrix Factorization
  - Gradient Descent algorithm
  - Regularization
  - Collaboration Filtering
- Non-negative Matrix Factorization