Roadmap

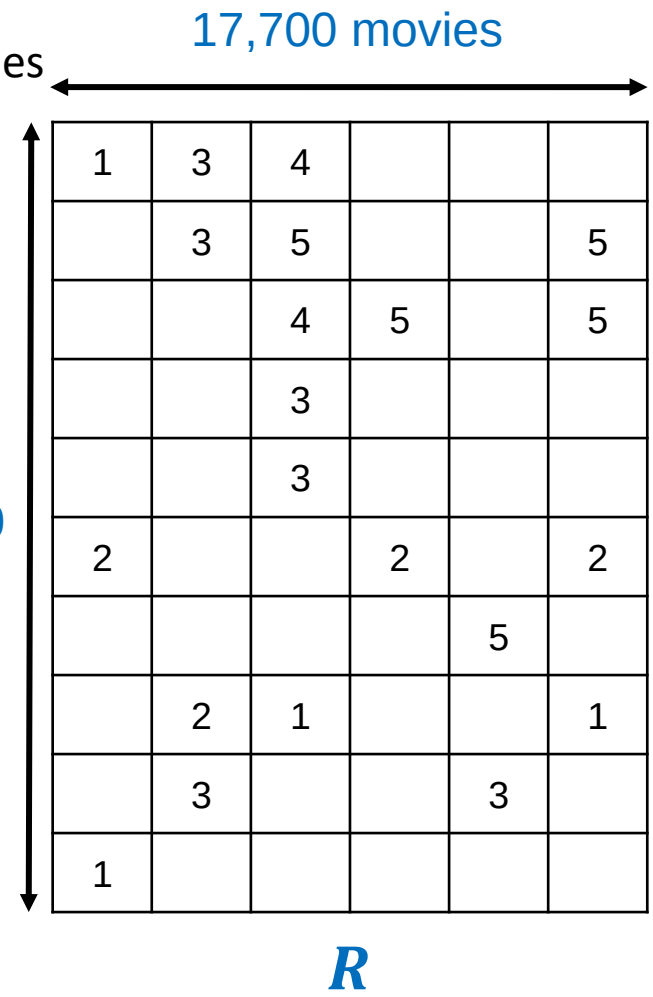
- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. Singular Value Decomposition (SVD)
  4. **Matrix Factorization**
     - **Motivation & Approach**
     - Regularization & Sparsity
     - Further Factorization Models
  5. Neighbor Graph Methods
  6. Autoencoders (Non-linear Dimensionality Reduction)

Motivation: The Netflix Prize

- Training data
  - 100 million ratings, 480,000 users, 17,770 movies
  - 6 years of data: 2000-2005
- Test data
  - Last few ratings of each user (2.8 million)
  - Root Mean Square Error (RMSE)
  - Netflix's system RMSE: 0.9514

trivial (avg): 1.054

- Competition
  - 2,700+ teams
  - $1 million prize for 10% improvement



Evaluating Recommender Systems

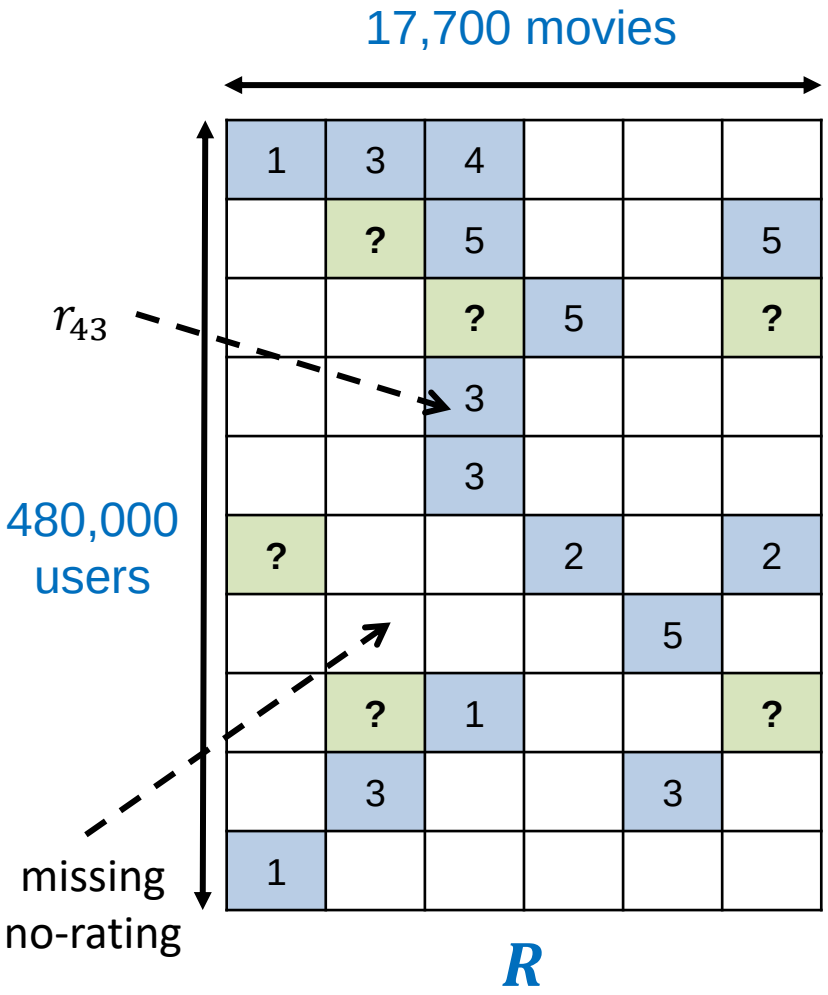
- S = set of tuples (u, i) of users u that have rated item i with a rating of r_{ui}

- $$\text{RMSE} = \frac{1}{|S|}\sqrt{\sum_{(u,i)\in S}(r_{ui} - \hat{r}_{ui})^2}$$

  r_{ui}: true rating of user u for item i

  $\hat{r}_{ui}$: predicted rating

- Legend:
  - Training and validation data
  - Test data
  - Missing data



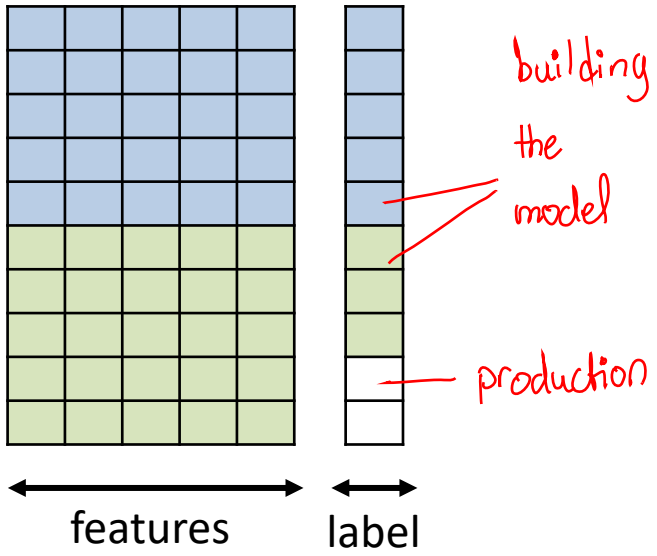
GOAL: predicting ↑

Regression vs. Recommendation

- Legend:
  - Training and validation data
  - Test data
  - Missing data

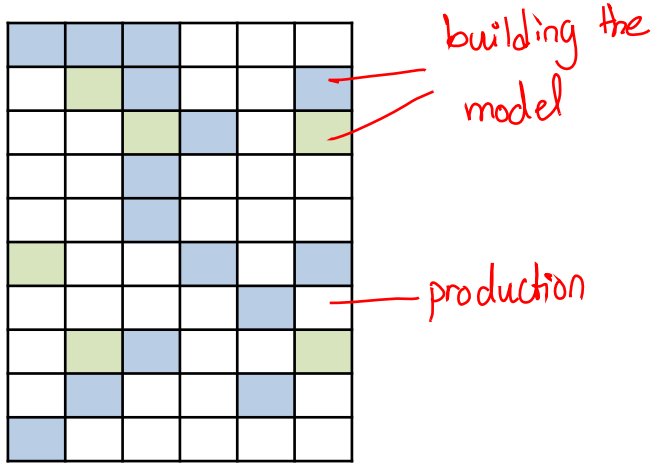
Matrix Completion

Regression



predicting labels for new users

Recommendation



predicting other ratings

not a single value

SVD on Rating Data

* When predicting missing data e.g 4, 2, 5
we choose the largest predicted score, 5

- Goal: Make good recommendations
  - Good performance on observed (user, item) ratings, i.e. low RMSE
  - Generalize to the unseen test data

* Can we use SVD to obtain the solution?
  - SVD on the rating matrix $\boldsymbol{R} \in \mathbb{R}^{n \times d}$ where we replace **missing** entries with **zeros**
  - $\boldsymbol{R} \approx \boldsymbol{Q} \cdot \boldsymbol{P}^T$ // SVD: $\boldsymbol{R} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \longrightarrow \boldsymbol{Q} = \boldsymbol{U}\boldsymbol{\Sigma}, \ \boldsymbol{P} = \boldsymbol{V}$

only keeping the largest K singular values

items

users

1	0	3	0	0	5	0	0	5	0	4	0
0	0	5	4	0	0	4	0	0	2	1	3
2	4	0	1	2	0	3	0	4	3	5	0
0	2	4	0	5	0	0	4	0	0	2	0
0	0	4	3	4	2	0	0	0	0	2	5
1	0	3	0	3	0	0	2	0	0	4	0

$\boldsymbol{R}$

$\approx$

factors

users

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

$\boldsymbol{Q}$

items

factors

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

$\boldsymbol{P}^T$

only priority for SVD

- SVD gives **minimum reconstruction error** (Sum of Squared Errors):

$$\min_{\boldsymbol{U},\boldsymbol{\Sigma},\boldsymbol{V}} \sum_{\substack{u=1\ldots n \\ i=1\ldots d}} \left(\boldsymbol{R}_{ui} - \left[\boldsymbol{U\Sigma V^T}\right]_{ui}\right)^2$$

B as optimal reconstruction of R

  - SSE and RMSE are monotonically related:
    - $\text{RMSE} = \frac{1}{\text{const.}}\sqrt{\text{SSE}}$
    - Great news: SVD is minimizing RMSE

- **Complication:**
  - The sum in the SVD error term is over **all** entries (no-rating = zero-rating)
  - But our $\boldsymbol{R}$ has **missing** entries!
  - (Classical) SVD isn't defined when entries are missing!
- Also: We actually don't care about orthogonality and normalization

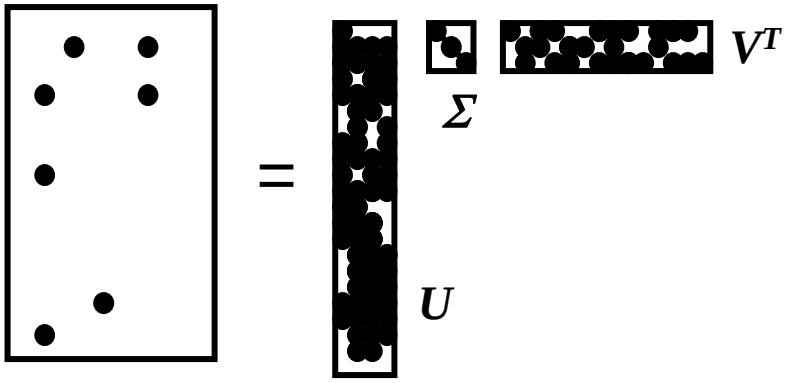
SVD is not ideal in this case!

Discussion: SVD/PCA

- Optimal low-rank approximation
  - In terms of Frobenius norm (and also in terms of the spectral norm)

- Missing elements (we have no information/not observed) are treated as 0 (a very low rating)
  - Critical for many applications (e.g. recommender systems)
  - General problem: two kinds of "zeros" (not observed/missing ≠ 0)

- Lack of Sparsity
  - Singular vectors are dense
  - Potential interpretability problem



- Orthogonality really required/useful?

* sparse factors are easier to interpret than dense factors
even though we aim for final sparse output

Latent Factor Models

- Our goal: Find $\boldsymbol{Q} \in \mathbb{R}^{n \times k}$ and $\boldsymbol{P} \in \mathbb{R}^{d \times k}$ such that:

$$\min_{\boldsymbol{P},\boldsymbol{Q}} \sum_{(u,i) \in S} \left(r_{ui} - \boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}}\right)^2$$

Unlike before, the zeros we added is not included

  - We only sum over existing entries, i.e. the set $S = \{(u,i) \mid r_{ui} \neq \text{missing}\}$)
  - We don't require columns of $\boldsymbol{P}, \boldsymbol{Q}$ to be orthogonal or unit length

- Use standard optimization techniques to solve this problem

scalar (1X1) = (1XK) · (KX1)

$$\hat{r}_{ui} = \boldsymbol{q}_u \cdot \boldsymbol{p}_i^T$$

$\boldsymbol{q}_u$ row u of $\boldsymbol{Q}$
$\boldsymbol{p}_i$ row i of $\boldsymbol{P}$

items

users

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

$\boldsymbol{R}$

$\approx$

factors

users

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

$\boldsymbol{Q}$

items

factors

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

$\boldsymbol{P^T}$

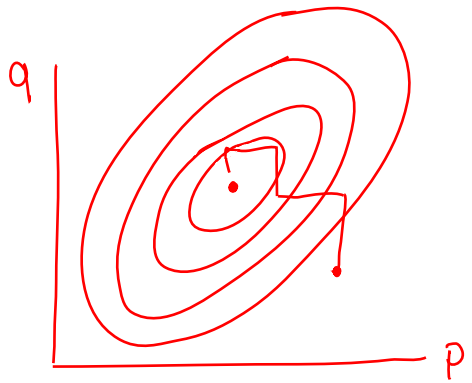
Each user
Each item
> has its own latent factor
"Embedding"
K-dim

Finding the Latent Factors (I)

- Goal: $\min_{\boldsymbol{P},\boldsymbol{Q}} \sum_{(u,i)\in S} \left(r_{ui} - \boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}}\right)^2 =: \min_{\boldsymbol{P},\boldsymbol{Q}} f(\boldsymbol{P},\boldsymbol{Q})$

- One approach: **Alternating Optimization** // a.k.a. block coordinate minimization
  - Pick initial values for $\boldsymbol{P}$ and $\boldsymbol{Q}$ think of each of P & Q as block of coord
  - Alternatingly keep one variable fix and optimize for the other
  - Repeat until convergence — error reached min / actual P & Q not changing by much

- Pseudo-Code:
  1. initialize $\boldsymbol{P}^{(0)}, \boldsymbol{Q}^{(0)}, t = 0$
  2. $\boldsymbol{P}^{(t+1)} = \mathrm{argmin}_{\boldsymbol{P}}\ f(\boldsymbol{P}, \boldsymbol{Q}^{(t)})$
  3. $\boldsymbol{Q}^{(t+1)} = \mathrm{argmin}_{\boldsymbol{Q}}\ f(\boldsymbol{P}^{(t+1)}, \boldsymbol{Q})$
  4. $t = t + 1$
  5. goto 2 until convergence



- Initialization of $\boldsymbol{P}$ and $\boldsymbol{Q}$:
  - Use, e.g., SVD where missing entries are replaced by 0 (or overall mean)

or any other heuristic

Alternatively, one can use random initialization

- How to solve $\boldsymbol{P}^{(t+1)} = \underset{\boldsymbol{P}}{\operatorname{argmin}} f(\boldsymbol{P}, \boldsymbol{Q}^{(t)}) = \underset{\boldsymbol{P}}{\operatorname{argmin}} \sum_{(u,i)\in S} \left(r_{ui} - \boldsymbol{q}_u \cdot \boldsymbol{p}_i^T\right)^2$

  constants; the only var
  - Observation 1: Since $\boldsymbol{Q}$ is fixed, the problem can be solved for **each** vector $\boldsymbol{p_i}$ **independently**

$$\min_{\boldsymbol{P}} \sum_{(u,i)\in S} \left(r_{ui} - \boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}}\right)^2 = \sum_{i=1\ldots d} \min_{\boldsymbol{p}_i} \sum_{u\in S_{*,i}} \left(r_{ui} - \boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}}\right)^2$$

where $S_{*,i} = \{u \mid (u,i) \in S\}$ // = only users u who have rated item i

  - Equivalently for vector $\boldsymbol{q}_u$ based on the set $S_{u,*} = \{i \mid (u,i) \in S\}$

* The 2 min are similar, however x_j is col vec; q_u, P_i^T are row vec

- *Observation 2:* $\min_{\boldsymbol{p}_i} \sum_{u \in S_{*,i}} \left(r_{ui} - \boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}}\right)^2$ is an **ordinary least squares regression** problem

convex obj fn

  - $\min_{\mathbf{w}} \sum_{j=1}^{n} \left(y_j - \boldsymbol{w}^T \boldsymbol{x}_j\right)^2$ (prediction) // y_j is scalar, $\boldsymbol{x}_j$ and $\boldsymbol{w}$ (column) vectors
  - Optimal solution in closed form: $\boldsymbol{w} = \left(\frac{1}{n}\sum_{j=1}^{n} \boldsymbol{x}_j \boldsymbol{x}_j^T\right)^{-1} \cdot \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{x}_j y_j$
    - ↳ taking grad wrt $w \overset{!}{=} 0$
    - $w = (X^T X)^{-1} X^T y$ matrix notation
  - In our case: $\boldsymbol{p}_i^T = \left(\frac{1}{|S_{*,i}|}\sum_{u \in S_{*,i}} \boldsymbol{q}_u^T \boldsymbol{q}_u\right)^{-1} \cdot \frac{1}{|S_{*,i}|}\sum_{u \in S_{*,i}} \boldsymbol{q}_u^T r_{ui}$
  - Equivalently for $\boldsymbol{q}_u$

- Computation of $\operatorname{argmin}_{\boldsymbol{P}}(\boldsymbol{P}, \boldsymbol{Q}^{(t)})$ reduces to a standard problem

Alternating Optimization: Discussion

i.e non convex

by decomposing them into small solvable optimization problems

- May provide solution to difficult optimization problems
- Here: sequence of simple OLS problems
  - Overall algorithm can be implemented in a few lines in, e.g., Python
  - Quite efficient: since data is sparse, the sets $S_{*,i}$ (and $S_{u,*}$) are relatively small

- **Drawback** of Alternating Optimization:
  - Solution is only an approximation
  - No guarantee that close to the optimal solution
  - Highly depends on initial solution

i.e might stuck with local optimum, when solving for each P & Q

SGD for Matrix Factorization

- Stochastic Gradient Descent as an alternative

- SGD on the objective $\mathcal{L} := \sum_{(u,i)\in S}\left(r_{ui} - \boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}}\right)^2$
  - Pick a random user u and a random item i with rating r_{ui} (batch size 1)
  - Compute the gradients w.r.t. the parameters $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_u}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{p}_i}$
  - Update the parameters $\boldsymbol{q}_u \leftarrow \boldsymbol{q}_u - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_u}$ $\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{p}_i}$

Step size is determinant by learning rate

- In this case the gradient update step is rather simple

SGD with batch size of 1

  - $e_{ui} \leftarrow r_{ui} - \boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}}$ \\ helper variable, the current error
  - $\boldsymbol{q}_u \leftarrow \boldsymbol{q}_u + 2\eta(e_{ui}\,\boldsymbol{p}_i)$
  - $\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + 2\eta(e_{ui}\,\boldsymbol{q}_u)$

* In case of larger batch size, grad update : < $\sum$ terms contributing to U / $\sum$ terms contributing to i

Rating Prediction

- How to estimate the missing rating of user u for item i?

items

users

1		3			5			5		4	
		5	4	?		4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

$\boldsymbol{R}$

$\approx$

largest K

$$\hat{r}_{ui} = \boldsymbol{q}_u \cdot \boldsymbol{p}_i^T = \sum_k q_{uk} \cdot p_{ik}$$

$\boldsymbol{q}_u$ row u of $\boldsymbol{Q}$

$\boldsymbol{p}_i$ row i of $\boldsymbol{P}$

latent factor

factors

users

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

$\boldsymbol{Q}$

$\bullet$

items

factors

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

$\boldsymbol{P^T}$

- How to estimate the missing rating of user u for item i?

items

users

1		3			5			5		4	
		5	4	?		4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

$\boldsymbol{R}$

$\approx$

$$\hat{r}_{ui} = \boldsymbol{q}_u \cdot \boldsymbol{p}_i^T = \sum_k q_{uk} \cdot p_{ik}$$

$\boldsymbol{q}_u$ row u of $\boldsymbol{Q}$
$\boldsymbol{p}_i$ row i of $\boldsymbol{P}$

factors

users

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

$\boldsymbol{Q}$

$\bullet$

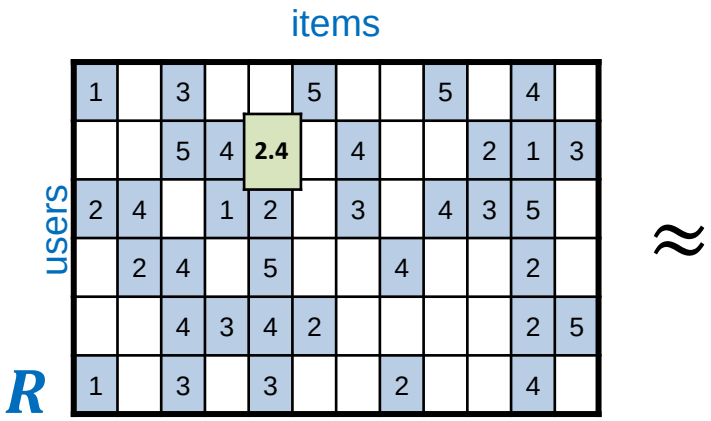
items

factors

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

$\boldsymbol{P^T}$

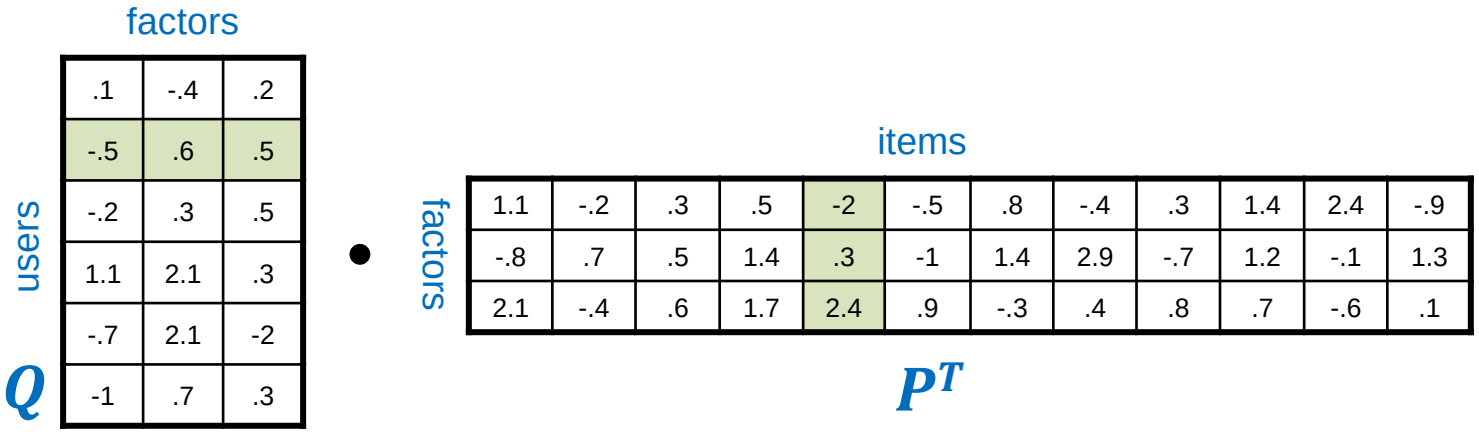
- How to estimate the missing rating of user u for item i?



$\approx$

$$\hat{r}_{ui} = \boldsymbol{q}_u \cdot \boldsymbol{p}_i^T = \sum_k q_{uk} \cdot p_{ik}$$

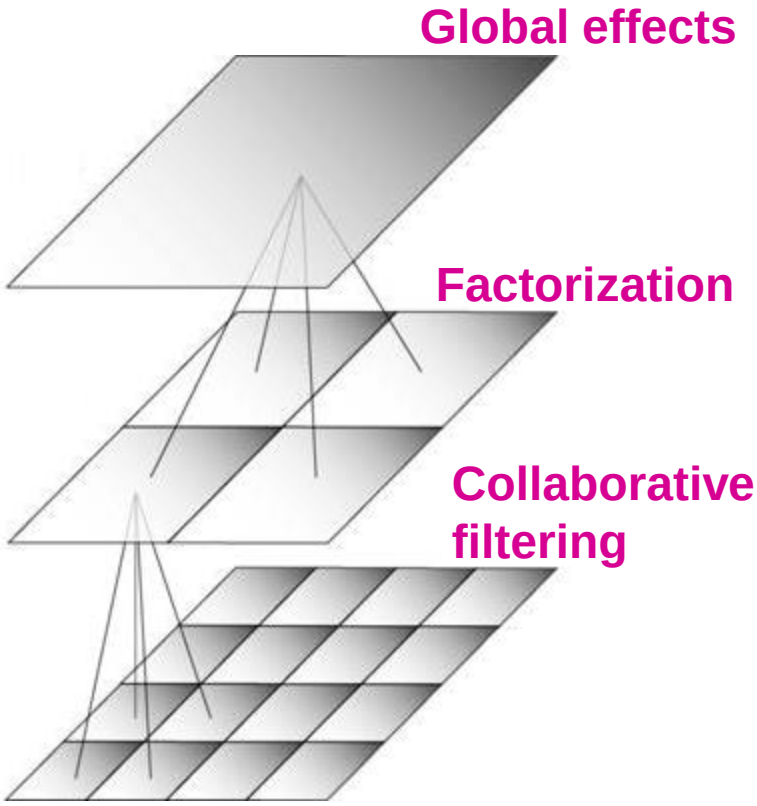
$\boldsymbol{q}_u$ row u of $\boldsymbol{Q}$
$\boldsymbol{p}_i$ row i of $\boldsymbol{P}$



Side-note: BellKor Recommender System

- The winner of the Netflix Challenge uses matrix factorization as **one** building block
  - The overall model is a combination of multiple ideas

- Multi-scale modeling of the data: Combine top level, "regional" modeling of the data, with a refined, local view:
  - **Global**:
    - Overall deviations of users/movies
  - **Factorization**:
    - Addressing "regional" effects
  - **Collaborative filtering**:
    - Extract local patterns



User and Item Biases

Global Effect Problem

- Certain users might give overly optimistic or pessimistic rating
- Certain movies tend to always have low ratings

- We introduce additional bias terms to capture these effects
  - Each user u can have a bias term b_u
  - Each item i can have a bias term b_i
  - Additional global bias term b shared by everyone

- The resulting optimization problem can be easily solved with SGD

$$\min_{\substack{\boldsymbol{P},\boldsymbol{Q} \\ b_u,b_i,b}} \sum_{(u,i)\in S} \left(r_{ui} - (\boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}} + b_u + b_i + b)\right)^2$$

- The **cold start** problem is another important issue

Having new movie, show it to random users

Similarly, having new user

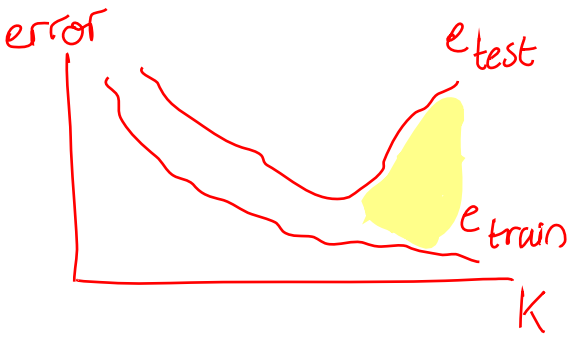
Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. Singular Value Decomposition (SVD)
  4. **Matrix Factorization**
     - Motivation & Approach
     - **Regularization & Sparsity**
     - Further Factorization Models
  5. Neighbor Graph Methods
  6. Autoencoders (Non-linear Dimensionality Reduction)

Challenge: Overfitting

* Our Key hyperparameter is
K: # latent factors i.e dim of $\langle q_i, p_i \rangle$

- Final Goal: Minimize SSE for **unseen test data**
- Proxy: Minimize SSE on **training data**
  - Want large k (number of factors) to capture all the signals
  - But, SSE on **test data** begins to rise for larger k
  - Why?



- Classical example of overfitting:
  - With too many degrees freedom (too many free parameters) the model starts fitting noise & many variations
  - The model fits too well the training data but does not generalize well to unseen test data

it becomes even worse with matrix factorization

- Problem can easily be seen in our scenario:
  - In each step of the alternating optimization we solve an OLS regression
    - Number of regression parameters = k = number of latent factors "latent size"
    - Number of data points used for regression = cardinality of $S_{*,i}$ / $S_{u,*}$
  - Lots of parameters but not enough data points → regression can overfit
  - If k is large this might even lead to an **underdetermined** system of equations
  - Problem is ill-posed

- Solution: **Regularization**
  - Interpretation for underdetermined systems: "In the absence of any other information, the parameter vector should be small (i.e. only small effect of features)"
  - Equivalently: We put a prior on the weights

Regularization

- To solve overfitting we introduce regularization:
  - Allow rich model where we have sufficient data
  - Shrink aggressively where data are scarce

penalizing $\|q_u\|$ $\|p_i\|$

$$\min_{P,Q} \sum_{(u,i)\in S} \left(r_{ui} - \boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}}\right)^2 + \left[\lambda_1 \sum_u \|\boldsymbol{q}_u\|^2 + \lambda_2 \sum_i \|\boldsymbol{p}_i\|^2\right]$$

"error" "length" the added regularization

  - λ_1 and λ_2 are user-defined regularization parameters

Note: We do not care about the actual value of the objective function, but we care about the $\boldsymbol{P}, \boldsymbol{Q}$ that achieve the minimum of the objective

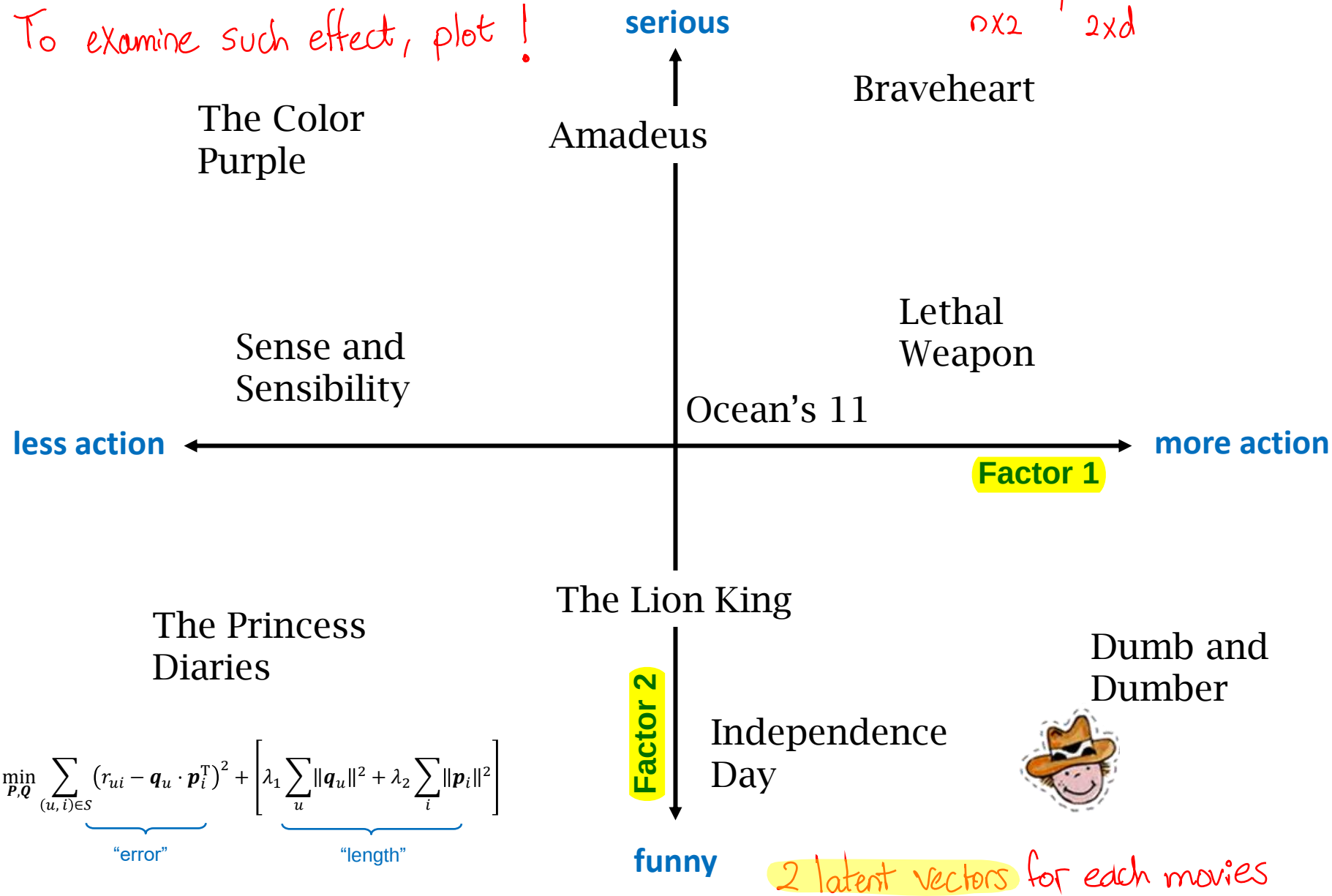
* Trade off bet how much you care about having: small norm / length; small error i.e good reconstruction

The Effect of Regularization

To examine such effect, plot !

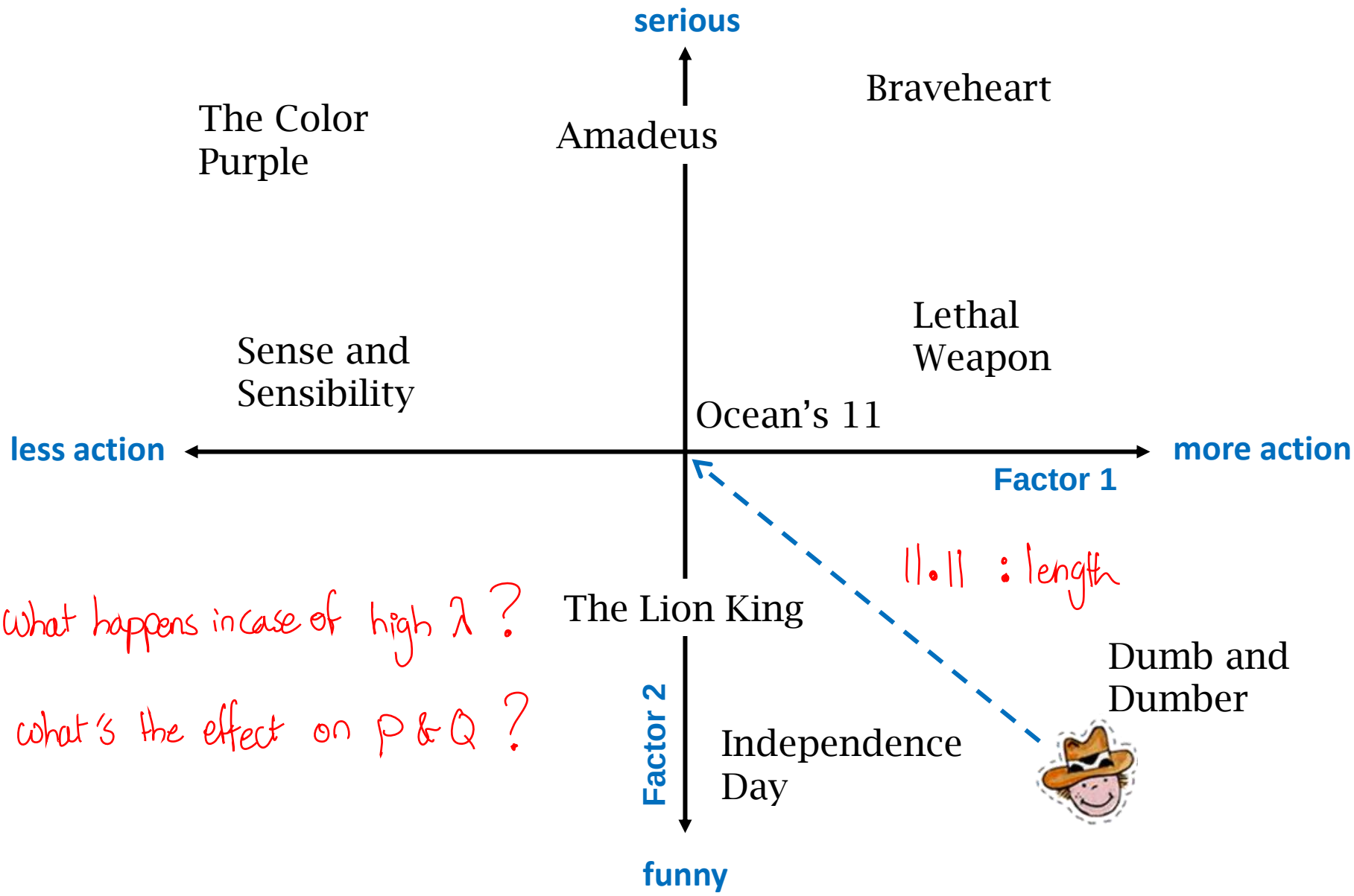
$R \approx Q \cdot P^T$

$n \times 2 \quad 2 \times d$



$$\min_{P,Q} \sum_{(u,i) \in S} \underbrace{(r_{ui} - \boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}})^2}_{\text{"error"}} + \underbrace{\left[\lambda_1 \sum_u \|\boldsymbol{q}_u\|^2 + \lambda_2 \sum_i \|\boldsymbol{p}_i\|^2\right]}_{\text{"length"}}$$

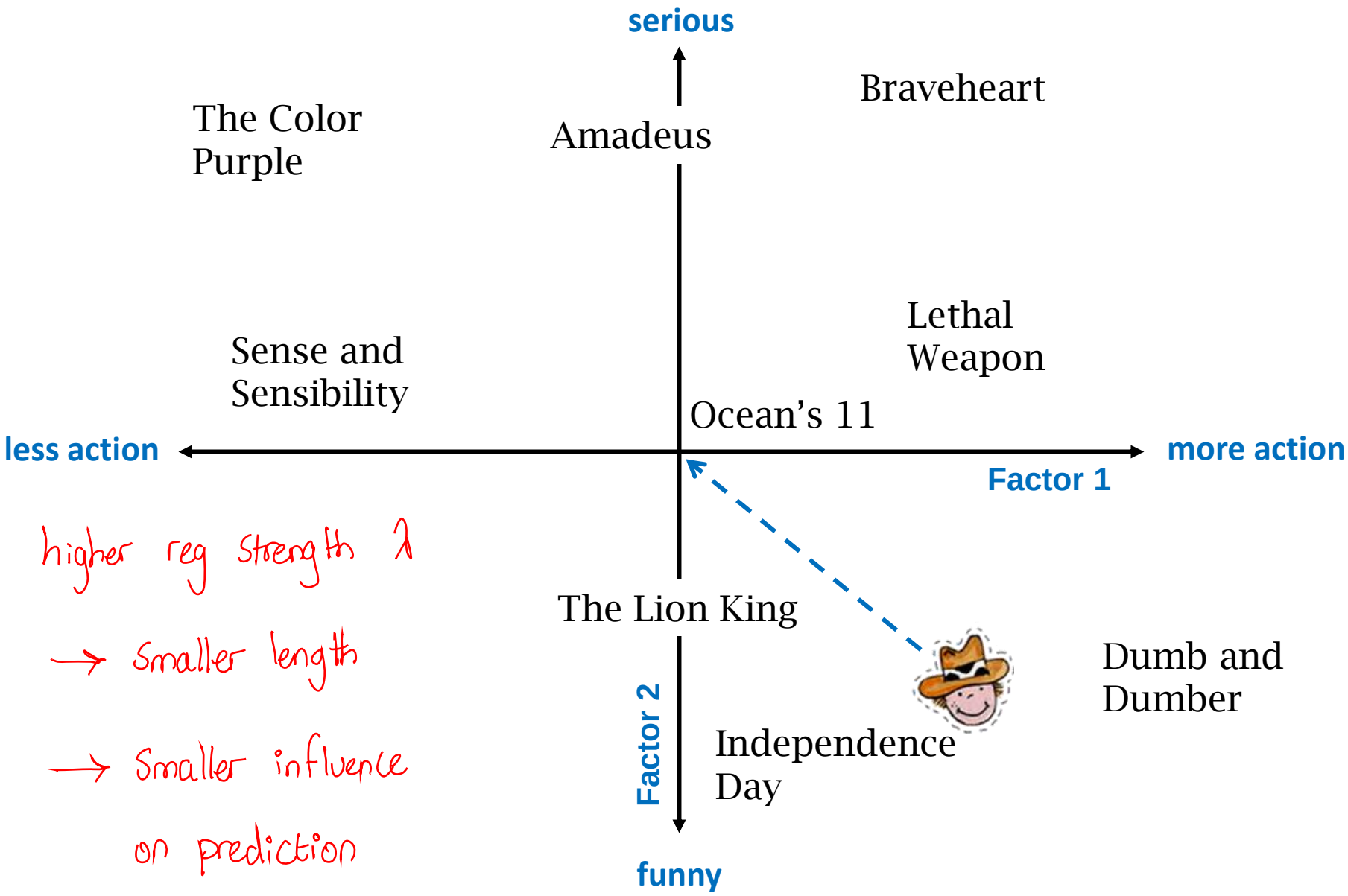
2 latent vectors for each movies

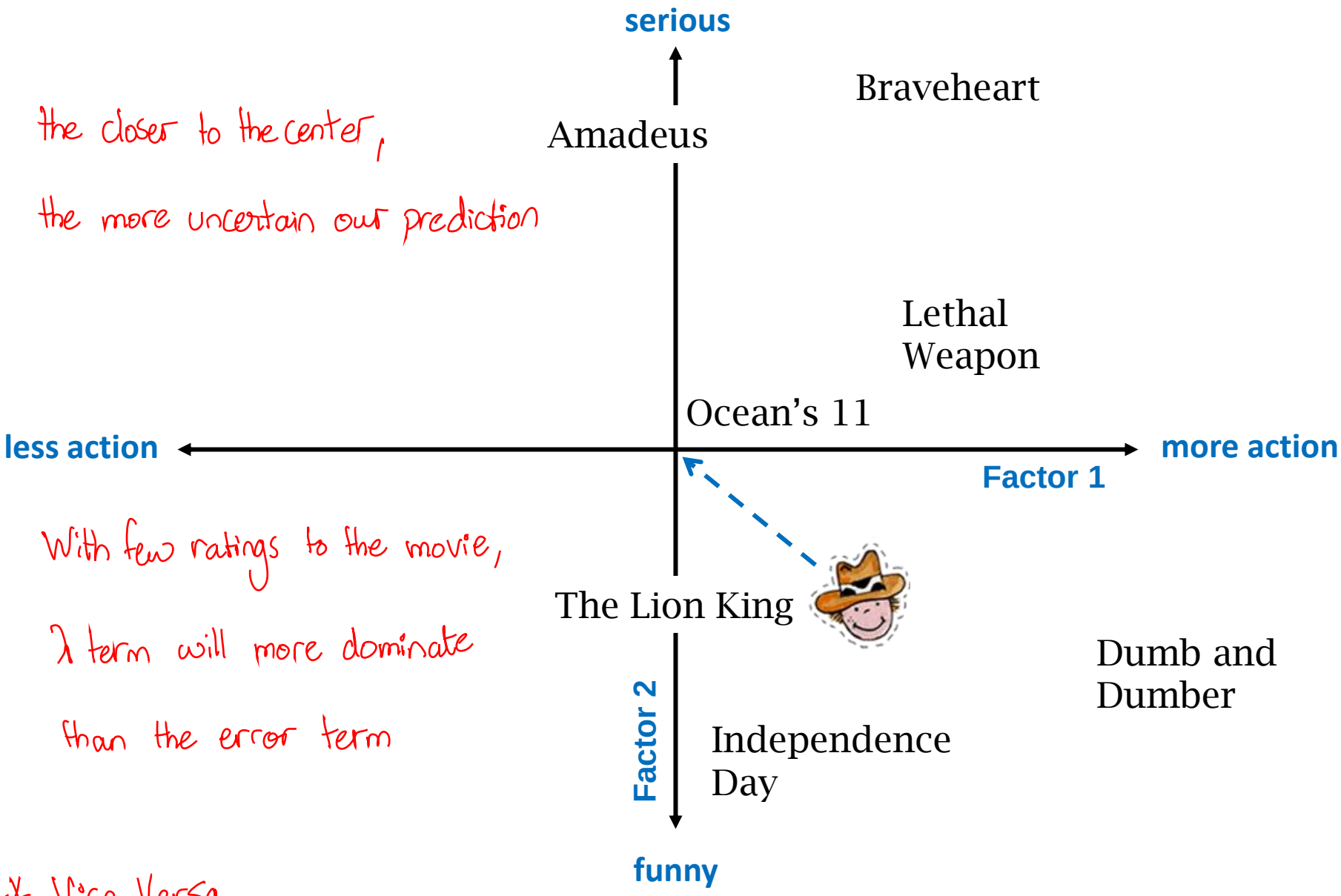


What happens incase of high λ ?

What's the effect on P & Q ?

||.|| : length


serious
Braveheart
The Color Purple
Amadeus
Lethal Weapon
Sense and Sensibility
Ocean's 11
less action
more action
Factor 1
higher reg strength λ
→ smaller length
→ smaller influence on prediction
The Lion King
Dumb and Dumber
Factor 2
Independence Day
funny



the closer to the center,

the more uncertain our prediction

With few ratings to the movie,

λ term will more dominate

than the error term

* Vice Versa

Regularization in Our Use Case

- Regularized Problem: $\min_{\boldsymbol{P},\boldsymbol{Q}} \sum_{(u,i)\in S} \left(r_{ui} - \boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}}\right)^2 + \left[\lambda_1 \sum_u \|\boldsymbol{q}_u\|^2 + \lambda_2 \sum_i \|\boldsymbol{p}_i\|^2\right]$

- Recap of unregularized problem: In each iteration of the alternating optimization we solved an OLS regression problem

each independently

- For the regularized version this becomes **ridge regression**
  - $\min_{\boldsymbol{p}_i} \sum_{i\in S_{*,i}} \left(r_{ui} - \boldsymbol{q}_u \cdot \boldsymbol{p}_i^T\right)^2 + \lambda_2 \|\boldsymbol{p}_i\|^2$

large λ →

  - Effect: parameter values are forced to become smaller
  - Large values that only capture noise are avoided

- You know how to solve this!
  - Closed form solution (see linear regression slides)
  - Gradient decent

$(X^T X + \lambda I)^{-1} X^T y$

SGD for MF with Regularization and Biases

Slide 13

- SGD on the objective

$$\mathcal{L} := \min_{\substack{\boldsymbol{P},\boldsymbol{Q} \\ b_u,b_i,b}} \sum_{(u,i)\in S} \left(r_{ui} - (\boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}} + b_u + b_i + b)\right)^2 + \left[\lambda_1 \sum_u \|\boldsymbol{q}_u\|^2 + \lambda_2 \sum_i \|\boldsymbol{p}_i\|^2\right]$$

- Pick a random user u and a random item i with rating r_{ui} (batch size 1)
- The gradient update step is very similar to before
  - $e_{ui} \leftarrow r_{ui} - (\boldsymbol{q}_u \cdot \boldsymbol{p}_i^{\mathrm{T}} + b_u + b_i + b)$ \\ helper variable, the current error
  - $\boldsymbol{q}_u \leftarrow \boldsymbol{q}_u + 2\eta(e_{ui}\,\boldsymbol{p}_i - \lambda_1 \boldsymbol{q}_u)$
  - $\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + 2\eta(e_{ui}\,\boldsymbol{q}_u - \lambda_2 \boldsymbol{p}_i)$
  - $b_u \leftarrow b_u + \eta e_{ui}$
  - $b_i \leftarrow b_i + \eta e_{ui}$
  - Usually directly set b to the global mean $b = \frac{1}{|S|}\sum_{(u,i)\in S} r_{ui}$

L2 vs. L1 Regularization

L_2 reg → adding prior with normal dist

L_1 reg → adding prior with log-loss dist

- Comparison: L2 vs. L1 regularization
  - L2 tries to "shrink" the parameter vector $\boldsymbol{w}$ **equally** in all directions
    - Large values are highly penalized due to the square in the L2 norm
    - Unlikely that any component will be **exactly 0**
  - L1 allows large values in individual components of $\boldsymbol{w}$ by shrinking other components to 0
  - L1 is suited to enforce **sparsity** of the parameter vector

$\|\cdot\|_2$ → soln not sparse

balancing each other out

NB

Why sparsity?

- Better interpretation
  - Only few values contribute to result
  - Unintuitive that **sparse input data** is generated based on **dense signal**
- Less storage, faster processing

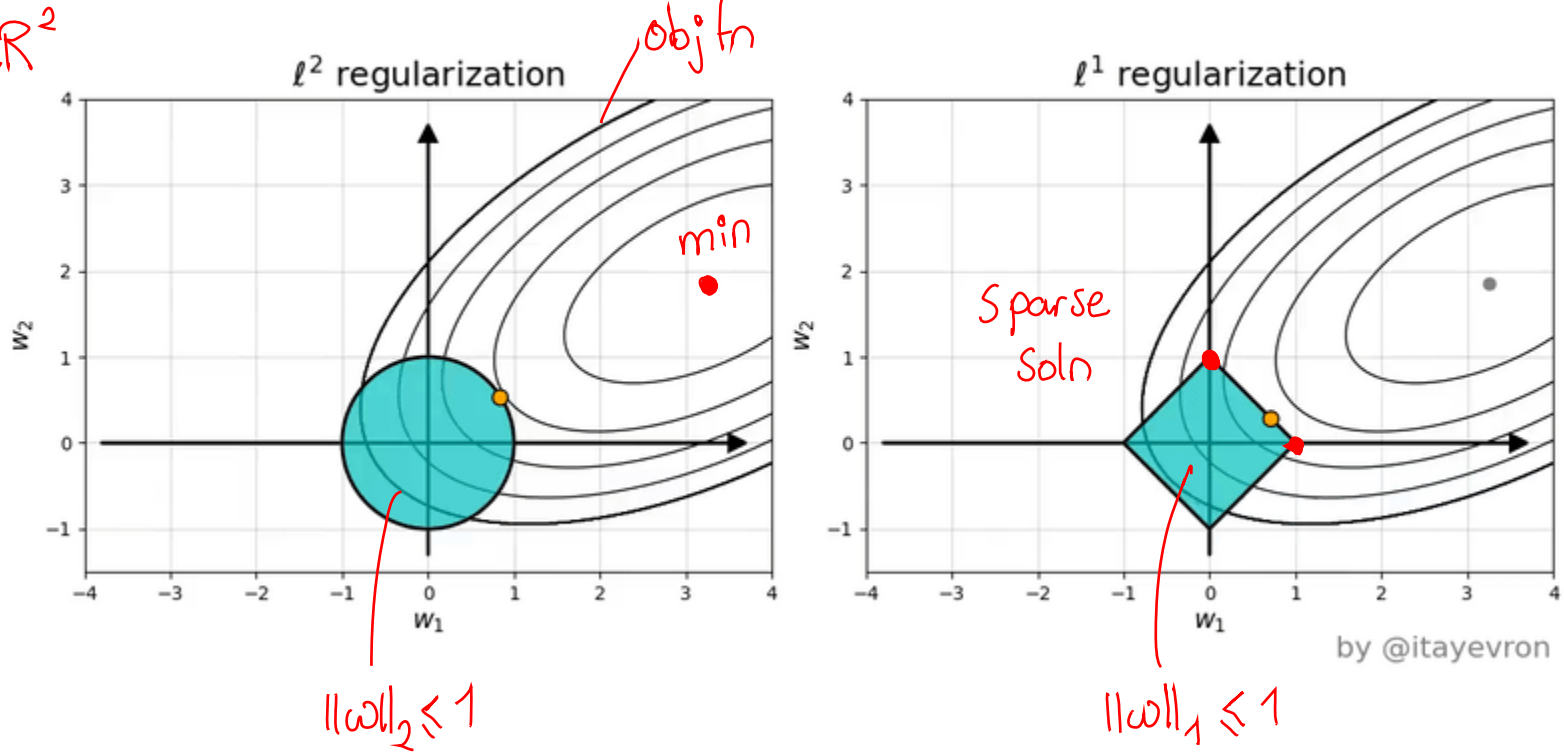
L2 vs. L1 Intuition

$\min_{w} f(w)$ $\quad \|w\| \leq 1$ $\xrightarrow{\text{dual}}$ $f(w) + \lambda(\|w\| - 1)$

- An L1 constraint promotes sparsity

$w \in \mathbb{R}^2$

here



* previously, it's about unconstrained optimization

here just for clarification, we used constrained

Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. Singular Value Decomposition (SVD)
  4. **Matrix Factorization**
     - Motivation & Approach
     - Regularization & Sparsity
     - **Further Factorization Models**
  5. Neighbor Graph Methods
  6. Autoencoders (Non-linear Dimensionality Reduction)

Further Factorization Models

- Matrix factorization is extremely powerful
  - Dimensionality reduction, data analysis/data understanding, prediction of missing values, …

Visualization → Compute 2/3 D latent factors, plot them & try to observe any hidden structure

- Various extensions have been proposed
  - Enforcing different constraints or operating on different data types
  - Important goal: better interpretation of result

- Non-Negative Matrix Factorization (next slides)
- Boolean Matrix Factorization
  - Factorize Boolean $\boldsymbol{A}$ in Boolean $\boldsymbol{Q}$ and $\boldsymbol{P}$

Non-Negative Matrix Factorization

- Often data is given in form of non-negative values
  - Rating values between 1 to 5; income, age, … of persons; number of words in a document; grayscale images; etc.

- However: SVD (and the other approaches presented before) might lead to factors containing **negative values**
  - Difficult to interpret: non-negative data is "generated" based on negative factors; what do these negative factors mean?
  - Predicted values might also become negative

- Solution: Non-Negative Matrix Factorization

i.e only learning the non-negative values

Non-Negative Matrix Factorization

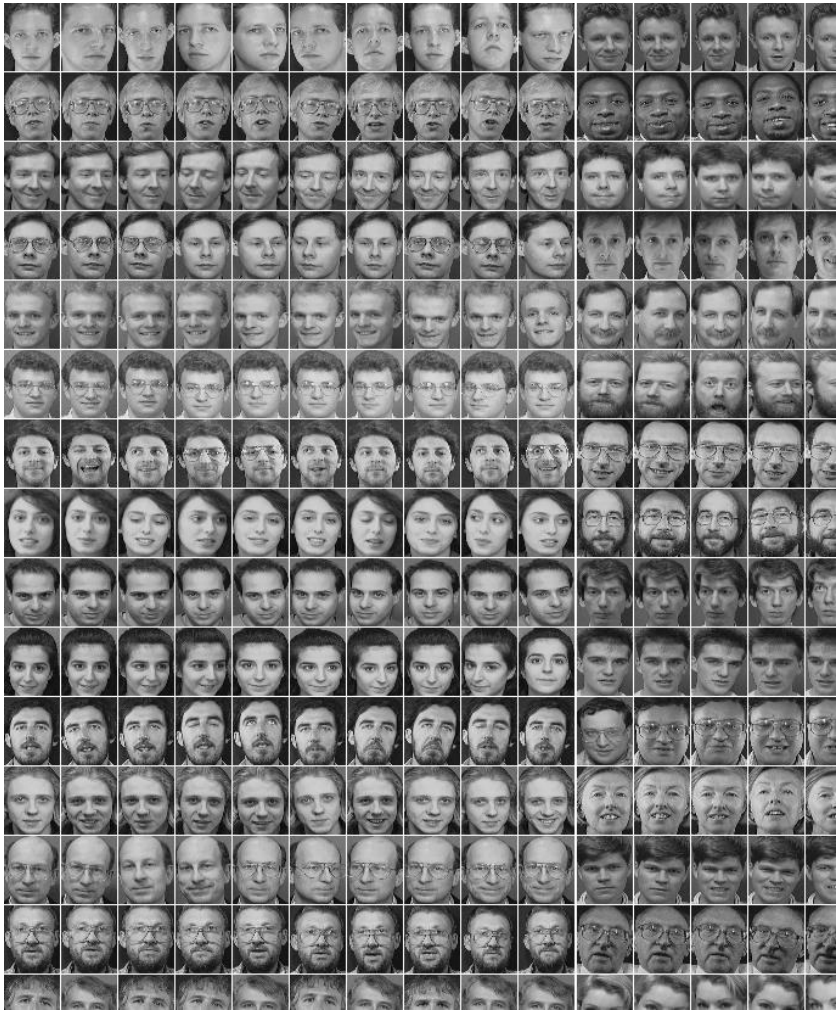
- Task: Factorize non-negative $\boldsymbol{A}$ in non-negative $\boldsymbol{Q}$ and $\boldsymbol{P}$, i.e. $\boldsymbol{A} \approx \boldsymbol{Q} \cdot \boldsymbol{P}^{\boldsymbol{T}}$

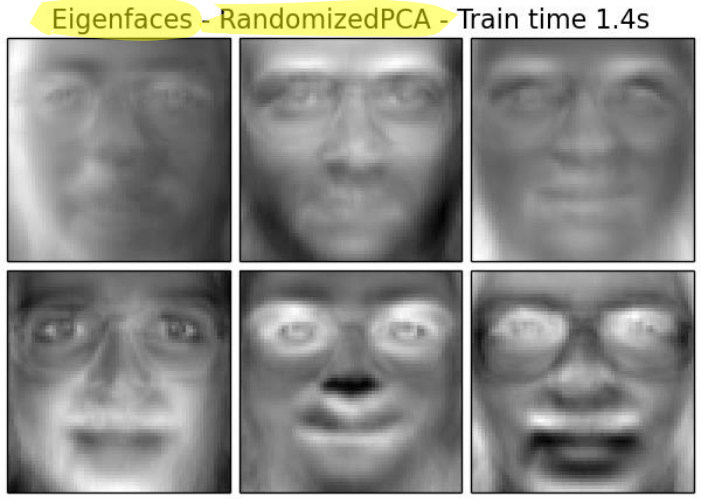
- Formally:
  - Given $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ with $\boldsymbol{A}_{ij} \geq 0$ and integer k, find $\boldsymbol{Q} \in \mathbb{R}^{n \times k}$, $\boldsymbol{P} \in \mathbb{R}^{d \times k}$ such that $\left\|\boldsymbol{A} - \boldsymbol{Q} \cdot \boldsymbol{P}^{\boldsymbol{T}}\right\|_F$ is minimized subject to $\boldsymbol{Q} \geq 0$ and $\boldsymbol{P} \geq 0$

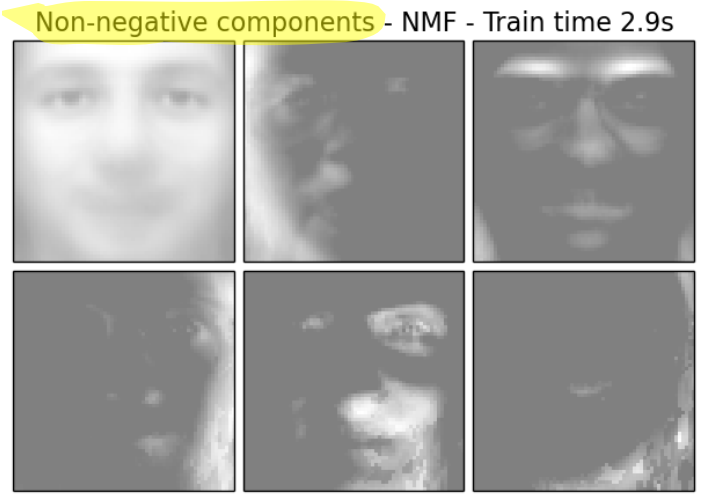
$$\min_{\boldsymbol{P} \geq 0, \boldsymbol{Q} \geq 0} \left\|\boldsymbol{A} - \boldsymbol{Q} \cdot \boldsymbol{P}^{\boldsymbol{T}}\right\|_F$$

  - Constrained optimization

NNMF Example: Olivetti Faces Data







results according to scikit-learn

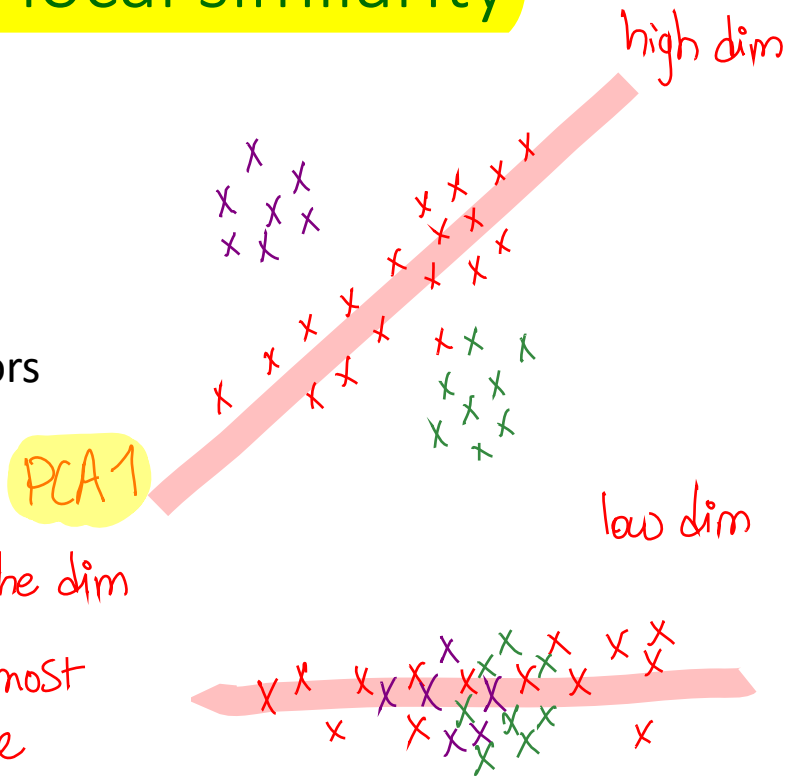
Roadmap

- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. Singular Value Decomposition (SVD)
  4. Matrix Factorization
  5. **Neighbor Graph Methods**
  6. Autoencoders (Non-linear Dimensionality Reduction)

Non linear Dim Reduction

Preserving global vs. preserving local similarity

- PCA tries to find a global structure
  - Can lead to local inconsistencies
  - Far away point can become nearest neighbors
- Illustration (during lecture):



- Idea: Preserve local structure instead

by looking at nearest neighbors

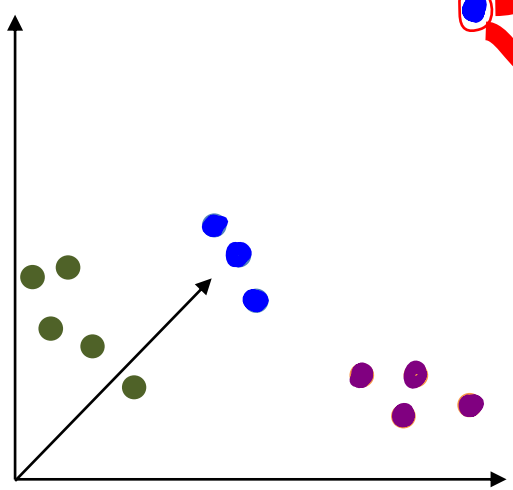
- Example: https://projector.tensorflow.org/

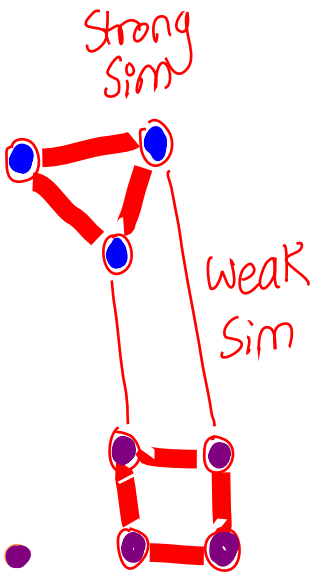
Neighbor Graph Methods

Matrix Factor. e.g PCA or SVD preserve global structure sometimes at the expense of local structure

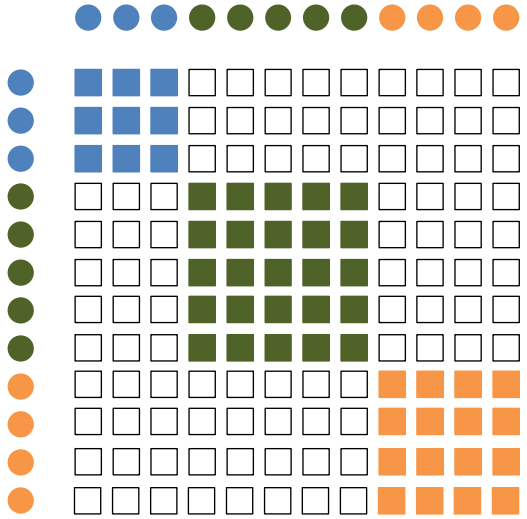
- All methods we have seen so far are based on **matrix factorizations**
- An alternative class of methods based on **neighbor graphs**

Highdimensional data:





Neighbor / similarity graph:



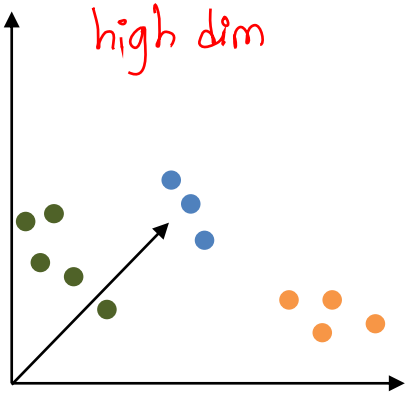
Idea: Low dimensional neighborhood similar to the original neighborhood

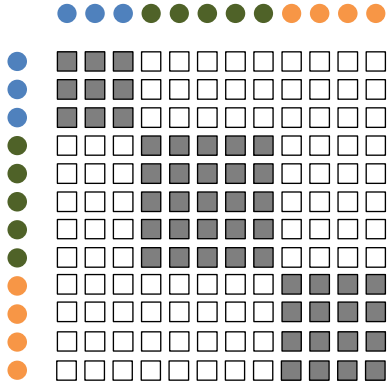
Neighbor Graph Methods

1. Construct neighbor graph of high-dimensional data
2. Initialize points in low-dimensional space
3. Optimize coordinates in low-dimensional space s.t. similarities align

high sim

low sim



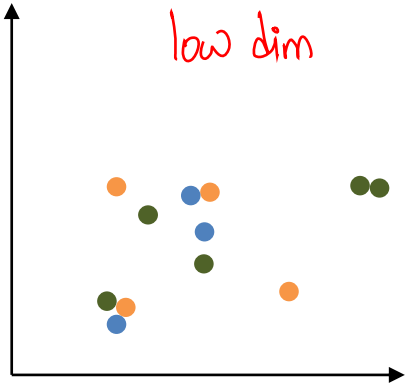


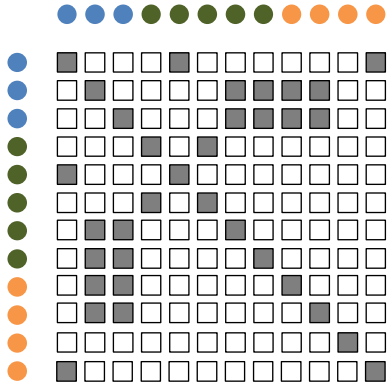
Adjacency Matrix

Pairwise Sim Matrix

Pairwise Sim Matrix

* We aim to move such randomly initialized data points, s.t the sim match those in high dim

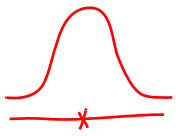




t-SNE: t-Distributed Stochastic Neighbor Embedding

- High-dimensional similarities for input $\boldsymbol{x}_i$:

apply Gaussian



$$p_{j|i} = \frac{\exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2}{2\sigma_i^2}\right)} \qquad p_{ii} = 0 \qquad p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

matrix rep how sim i & j

  - σ_i chosen to achieve fixed perplexity (effective number of neighbors)

- Low-dimensional similarities for (to be optimized) parameters $\boldsymbol{y}_i$:

apply t-dist. with 1 dof

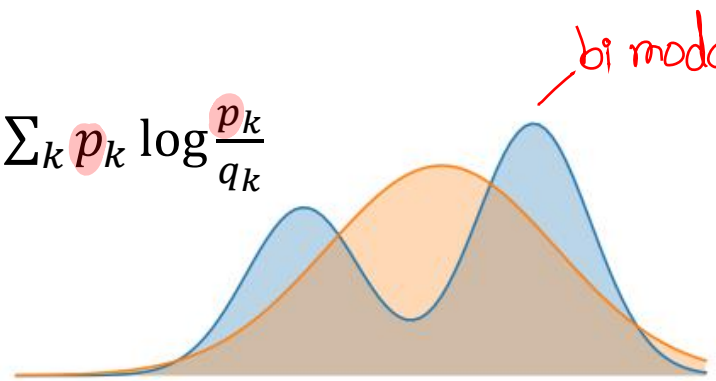
$$q_{ij} = \frac{\left(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2\right)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}} \qquad q_{ii} = 0$$

- Minimize the KL divergence: $\min_{\boldsymbol{y}_i} KL(P||Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$

Note on the KL Divergence

- $KL(P||Q) \geq 0$ for any P and Q, $KL(P||Q) = 0$ iff $P = Q$
- The KL Divergence is asymmetric $KL(P||Q) \neq KL(\mathrm{Q}||\mathrm{P})$
  - Example: Given P we are optimizing over Q

- $KL(P||Q) = \sum_k p_k \log \frac{p_k}{q_k}$

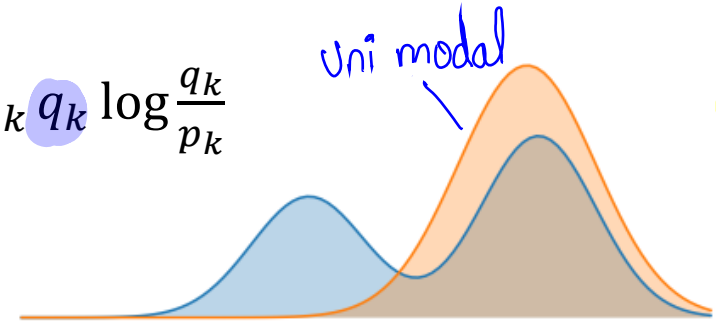


Forward KL is Mean-Seeking

$KL(P||Q)$

the one we choose

- $KL(\mathrm{Q}||\mathrm{P}) = \sum_k q_k \log \frac{q_k}{p_k}$



Reverse KL is Mode-Seeking

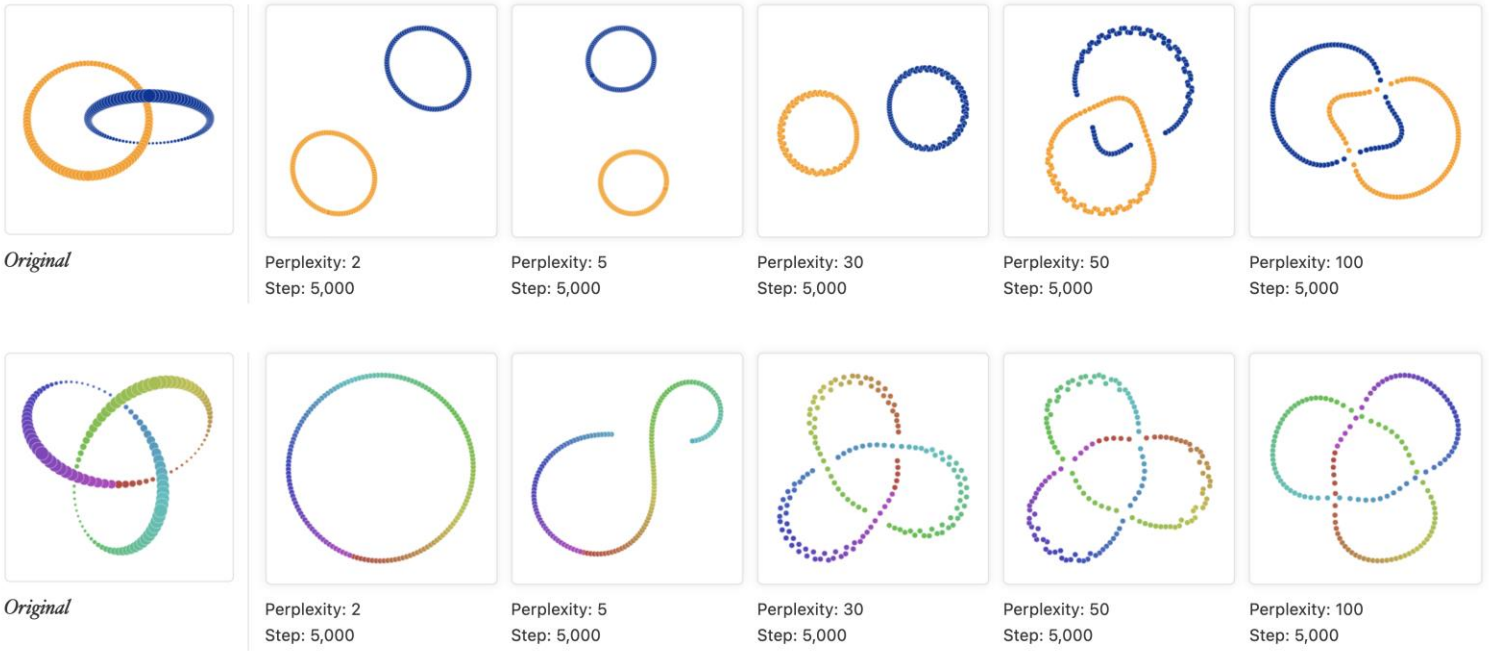
$KL(\mathrm{Q}||\mathrm{P})$

Figures adapted from Dibya Ghosh.

visualization is for cont dist

Interactive t-SNE

NB Cluster sizes & distances bet clusters does not have any intrinsic meaning by their own!

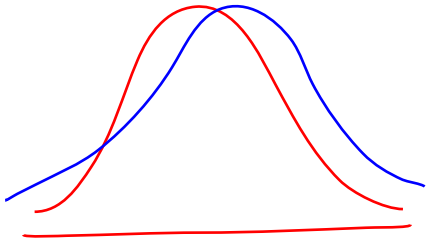


Source of illustration, for more details and interactive widgets.

The Advantages and Disadvantages of t-SNE

- The current standard for visualizing high-dimensional data
- Helps understand "black-box" algorithms like DNN
- Reduced "crowding problem" with heavy tailed distribution

the reason of choosing
- Gaussian for high dim
- t-dist for low dim



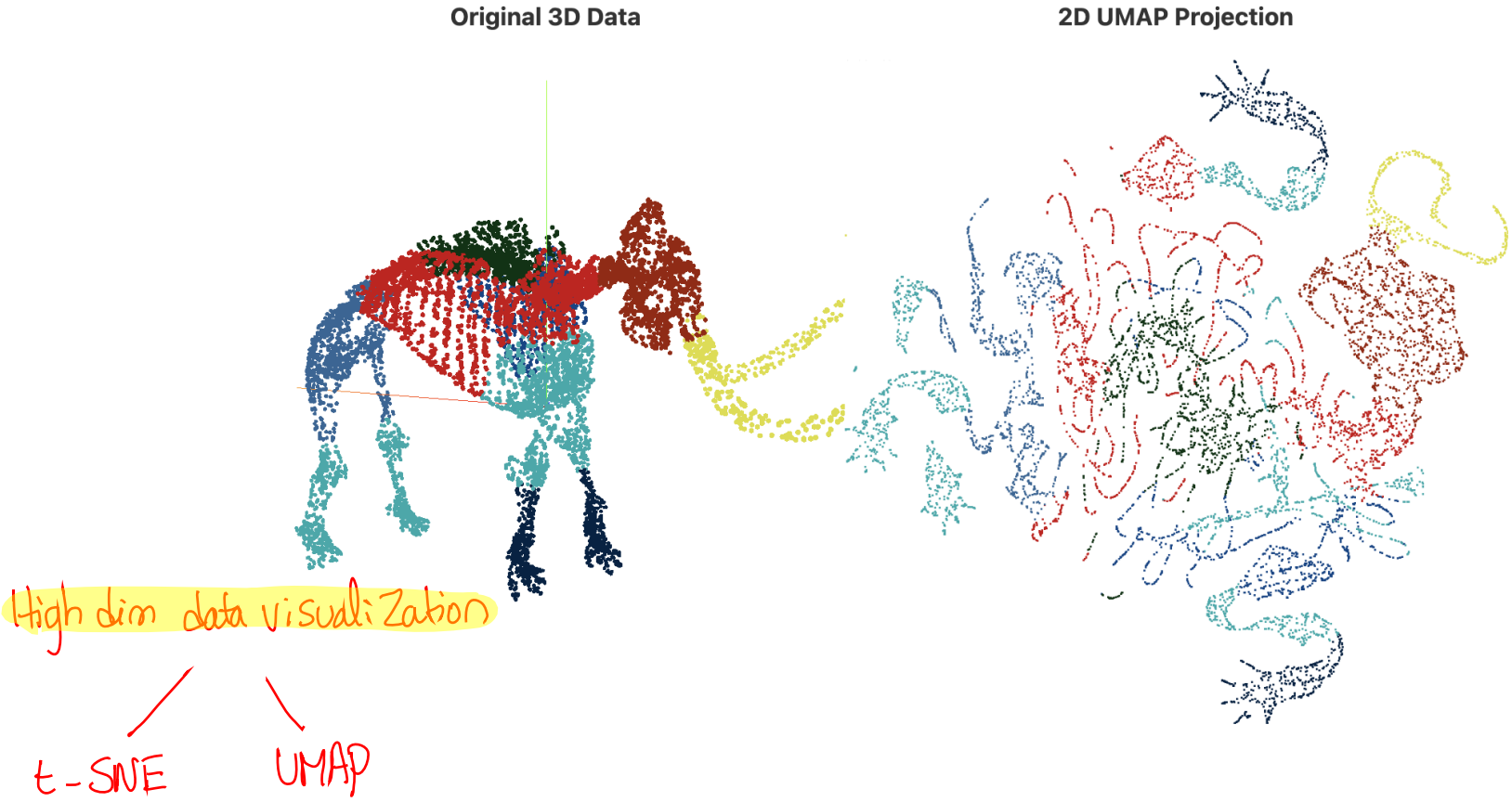
fatter tails → allowing points to be farther apart

- t-SNE plots can sometimes be mysterious or misleading
  - Be careful with cluster sizes and cluster distances
- Can be sensitive to hyperparameters
  - Random noise does not always look random
- No easy way to compute the embedding of new data
- Not great for more than 3 dimensions

mostly visualization, not dim reduction

Uniform Manifold Approximation and Projection

- Principled approach relying on toplogical spaces, category theory, …



Link for details, widgets, a comparison to T-SNE, …

Roadmap

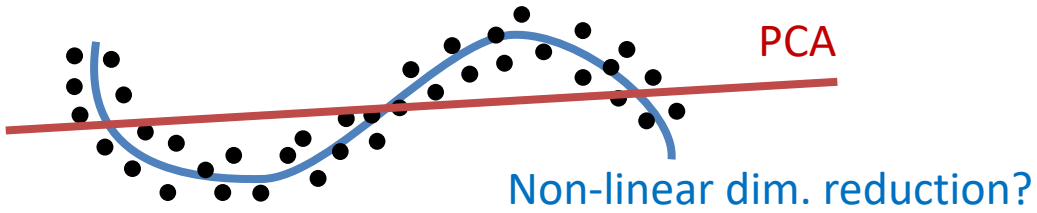
- Chapter: Dimensionality Reduction & Matrix Factorization
  1. Introduction
  2. Principal Component Analysis (PCA)
  3. Singular Value Decomposition (SVD)
  4. Matrix Factorization
  5. Neighbor Graph Methods
  6. **Autoencoders (Non-linear Dimensionality Reduction)**

Motivation

- PCA / SVD can only capture linear structure (linear variations in the data)
  - Recall: transformed data is $\boldsymbol{Y} = \widetilde{\mathbf{X}} \cdot \boldsymbol{\Gamma}$
  - Linear projection by the eigenvectors $\boldsymbol{\Gamma}$

linear transformation, capturing linear variations only

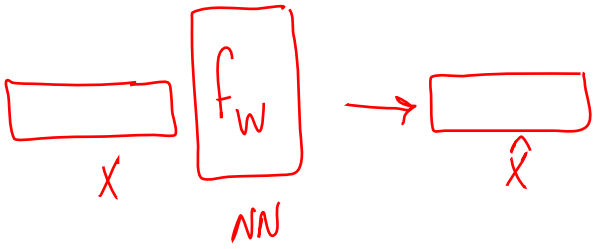
- However, data often lies on a non-linear low-dimensional manifold



- Idea: find a non-linear projection of the data

NB NN mainly aim to learn identity

Autoencoders



- An **autoencoder** is a neural network that:
  - finds a compact representation of the data
  - by learning to reconstruct its input

reconstructed input

$$f(\boldsymbol{x}, \boldsymbol{W}) := \hat{\boldsymbol{x}} \approx \boldsymbol{x}$$

→ does not depend linearly on x, but in non-linear fashion implemented by NN

- Goal: minimize the reconstruction error between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$

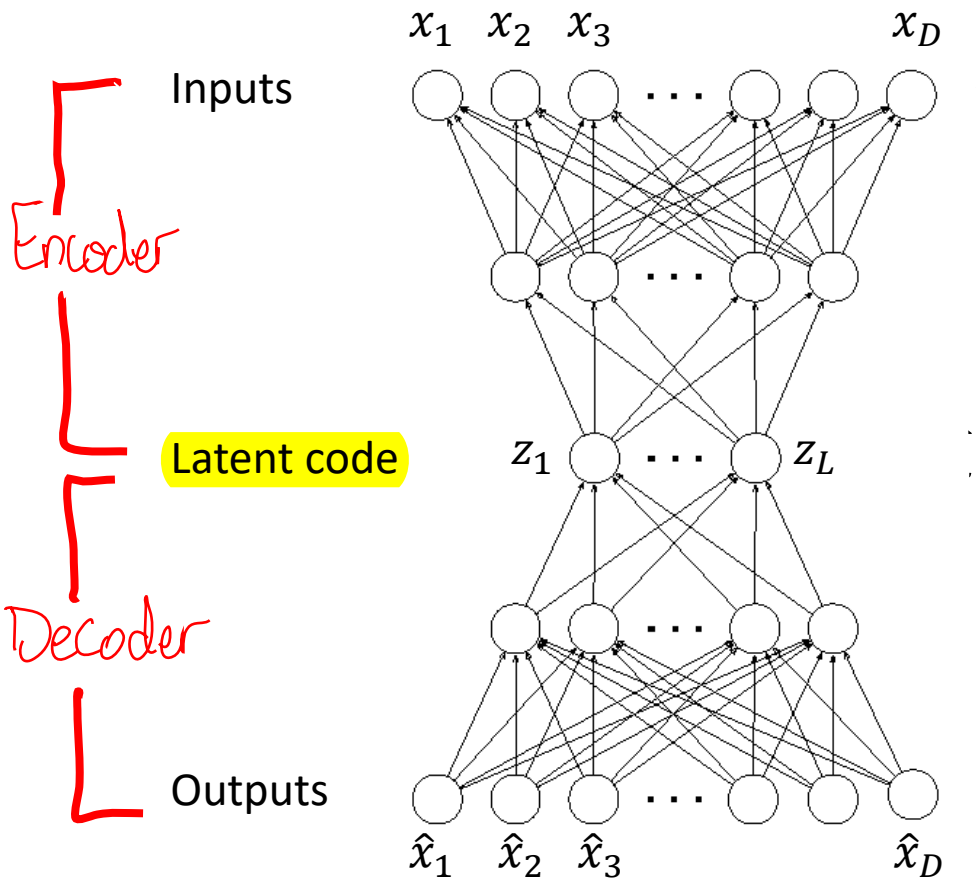
$$\min_{\boldsymbol{W}} \frac{1}{N} \sum_{i=1}^{N} \|f(\boldsymbol{x}_i, \boldsymbol{W}) - \boldsymbol{x}_i\|^2$$

- Alternative view: find a **latent representation** $\boldsymbol{z} \in \mathbb{R}^L$ which is a compact code for $\boldsymbol{x} \in \mathbb{R}^D$ since $L \ll D$
  - $f_{enc}(\boldsymbol{x}) = \mathbf{z}$ # encoder: project the data to a lower dimension
  - $f_{dec}(\boldsymbol{z}) \approx \boldsymbol{x}$ # decoder: reconstruct the data from the latent code

$$f_{dec}\big(f_{enc}(\boldsymbol{x})\big) \approx \boldsymbol{x}$$

approximate, not exact

- f_{enc} and f_{dec} are non-linear functions implemented by a neural network



① filters out unnecessary noise

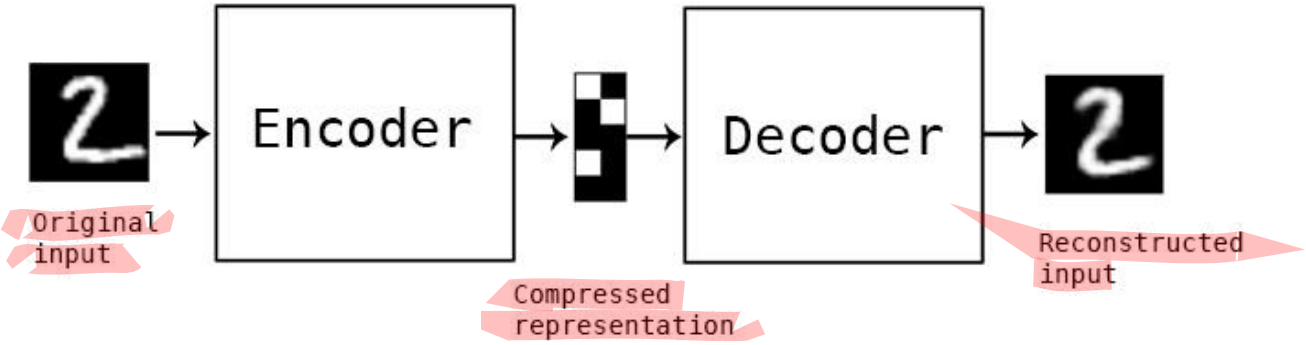
② preventing NN from learning the identity, since it will be useless

③ serving as low-dim rep of input

Bottleneck ($L \ll D$):
middle layer has fewer neurons

Autoencoders: Example

Check notes!



Original image:

Reconstruction:

- An autoencoder whose code dimension is less than the input dimension ($L \ll D$) is called **undercomplete**
  - Forces the autoencoder to capture the most salient features of the data
  - $L \geq D$ (**overcomplete**) the autoencoder can simply learn the identity function

- Training autoencoders in practice:
  - Add a regularization term to the SSE loss to prevent overfitting
  - Weight tying: f_{enc} and f_{dec} share the same weights

Sometimes leads to better performance;
Since # parameters reduced by half

- Other extensions:
  - Denoising autoencoders (DAEs): receive a corrupted (noisy) training data point as input and predict the "clean" uncorrupted data point as output
  - Variational autoencoders (covered in our MLGS lecture)

noisy clean

$$\| f(\tilde{x}_i, W) - x_i \|^2$$

(Linear) Autoencoders & PCA: Comparison

- What if f_{enc} and f_{dec} are linear functions?
  - $f_{enc}(\boldsymbol{x}, \boldsymbol{W}_1) = \boldsymbol{x}\boldsymbol{W}_1$ # $\boldsymbol{W}_1 \in \mathbb{R}^{D \times L}$
  - $f_{dec}(\boldsymbol{z}, \boldsymbol{W}_2) = \boldsymbol{z}\boldsymbol{W}_2$ # $\boldsymbol{W}_2 \in \mathbb{R}^{L \times D}$

- We have: $f_{dec}(f_{enc}(\boldsymbol{x})) = \boldsymbol{x}\boldsymbol{W}_1\boldsymbol{W}_2$ and

training linear autoencoders can be seen as PCA

$$\min_{\boldsymbol{W}_1\boldsymbol{W}_2} \frac{1}{N}\sum_{i=1}^{N} \|f(\boldsymbol{x}_i, \boldsymbol{W}) - \boldsymbol{x}_i\|^2 = \min_{\boldsymbol{W}_1\boldsymbol{W}_2} \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{x}_i\boldsymbol{W}_1\boldsymbol{W}_2 - \boldsymbol{x}_i\|^2 = \min_{\boldsymbol{W}} \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{x}_i\boldsymbol{W} - \boldsymbol{x}_i\|^2$$

Equivalent: $\boldsymbol{W}_1\boldsymbol{W}_2 = \boldsymbol{W}$ s.t. rank of $\boldsymbol{W}$ is L

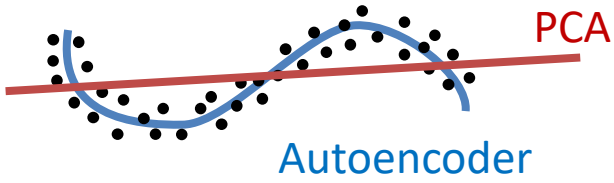
Collapsing into

- Optimal solution (assuming normalization):
  - PCA: $\boldsymbol{W}^* = \boldsymbol{\Gamma}$ where $\boldsymbol{\Gamma}$ are the (top-L) eigenvectors of $\boldsymbol{X}^T\boldsymbol{X}$

Check notes

Summary

- Dimensionality Reduction and Matrix Factorization are highly related
  - And can be used for various purposes
- PCA, SVD (and linear Autoencoders) are equivalent
  - Optimal low-rank approximation
- Matrix factorization for rating prediction
  - Very general formulation: allows to handle, e.g., missing entries
- Autoencoders and t-SNE perform non-linear dimensionality reduction
- Why are these techniques useful?
  - Less storage required
  - More efficient processing of the data
  - Remove redundant and noisy features
  - Discover hidden correlations/topics/concepts
  - Interpretation and visualization



* t-SNE is not helpful when images are translated, since the Euclidean dist will be large.

Autoencoder with CNN is better suited here!

Reading material

* t-SNE captures local structure.
* Autoencoder is adaptive to diff data types.

Reading material

- Bishop, chapters: 12.1, 12.2.1
- Leskovec, Rajaraman, Ullman - Mining of Massive Datasets: chapter 11
- Goodfellow - Deep Learning: chapter 14
- t-SNE
- Understanding UMAP