Recommender Systems

Sewoong Oh

CSE/STAT 416
University of Washington

Personalization is a successful use of learning from data

- Facebook advertisements from browsing history
- Amazon, YouTube, Netflix recommendations from user choices
- Input: user preferences (or activities)
- Goal: select (a small set of) items the user will like

					
User 1	5				3
User 2		2		4	
User 3			3		
User 4	1				
User 5			4		
User 6		5			2

- Challenge: sparsity
- Key idea: **collaborative filtering**
  a user might like something, if similar users liked it

Challenges in recommender systems

- Some feedback are implicit
  - explicit feedback: rating, purchase history, ranking
  - Implicit feedback: browsing history, TV viewing pattern
    - Implicit feedback requires pre-processing of data such as time spent, clicked, interval, etc.

- We seek diversity which is not easy to impose because users are multifaceted
  - A person with Linear Algebra textbook does not need another one, but Top-k recommendation might stick to k linear algebra textbooks
  - We don't want to recommend just Marvel movies

- Cold-start is hard
  - Recommendations for new user/movie with no data is hard
  - Need to use additional features/contexts (Netflix 20 questions)

Challenges in recommender systems

- Interests change over time, but dynamic models are hard to train
  - Users preferences change over time
  - Movies perception changes over time

- Given millions of users and hundreds of thousands of movies, we need a scalable (i.e. fast) algorithm
  - We need to exploit that data is sparse

Approach 0: popularity

- No personalization
- Netflix: trending now (average number of viewers)
- NY times: popular article (average views)

Approach 1: classifier

- Train a classifier on
  x = (user features and movie features)
  y = liked (+1) or not (-1)
- Output: +1 (recommend) or -1 (do not recommend)
- Pros
  - personalized
  - flexible to include additional features like time
- Cons
  - Useful features are hard to get
  - Empirical performance not as good as **Collaborative Filtering**

Approach 2: co-occurrences

- “People who bought X also bought …”
- Construct a normalized co-occurrence matrix C where both rows and columns indicate items

$$C_{ij} = \frac{\#\text{ of people who bought } i \text{ and } j}{\#\text{ of people who bought } i \text{ or } j}$$

- This is a symmetric matrix: $C_{ij} = C_{ji}$

- For a user who bought {milk, diapers}

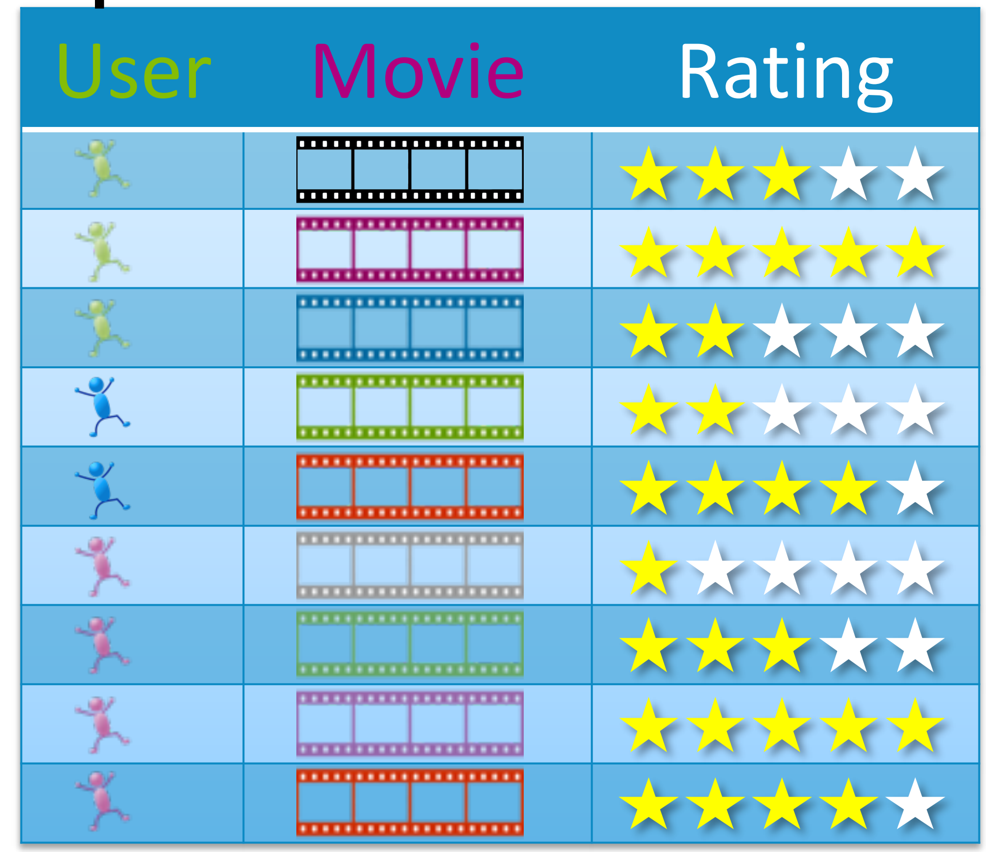
$$\text{Score}(\text{baby wipes}, user) = \frac{C_{\text{baby wipes, milk}} + C_{\text{baby wipes, diapers}}}{2}$$

- If user bought similar items, then give more score for “baby wipes”

Approach 3: matrix factorization

- Movie recommendations
- Users watch movies and give ratings
- But each user only rates a few

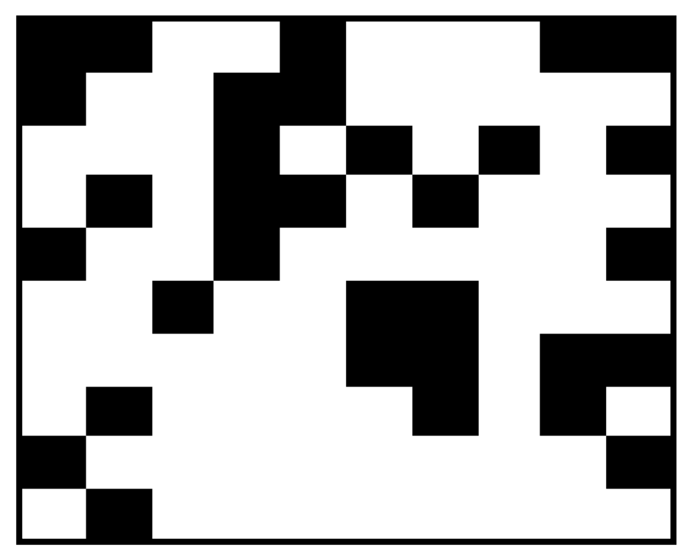
Input Data



Input Data in a matrix form

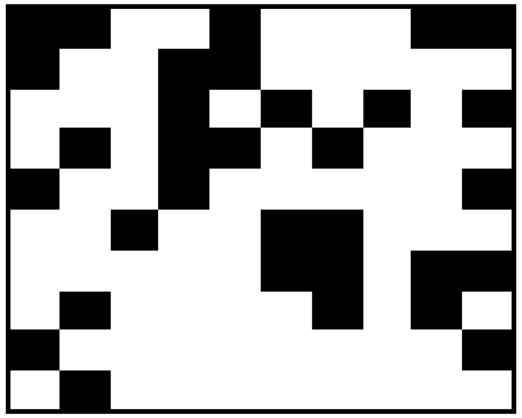
User 1	5				3
User 2		2		4	
User 3			3		
User 4	1				
User 5			4		
User 6		5			2

=



Matrix completion problem

Rating =



- Black cells indicate Rating(user,movie) known
- White cells indicate Rating(user,movie) unknown
- Each cell has values in {1,2,3,4,5}
- Goal: predict missing entries

Premise: Suppose we have *d* types of movies

- We can describe each movie ***v*** with feature vector $\boldsymbol{R_v}$
  - How much is the movie *action, romance, drama, …*
  - $\boldsymbol{R_v}$ = [0.3, 0.01, 1.5, …]
- We can describe a user ***u*** with feature vector $\boldsymbol{L_u}$
  - How much she likes *action, romance, drama, …*
  - $\boldsymbol{L_u}$ = [2.3, 0 , 0.7, …]
- Perhaps we can find such features that the rating can be predicted as the **inner product** of those two vectors

- Rating(u,v) = 0.3*2.3 + 0.01*0 + 1.5*0.7 + …

- This allows you to predict how a user will rate a movie, that she has not seen yet

Product recommendations

- Suppose the following features have been learned, which movie should we recommend to user #3?

User ID	Feature
1	(2, 0)
2	(1, 1)
3	(0, 1)
4	(2, 1)

Call this 4x2 matrix *L*

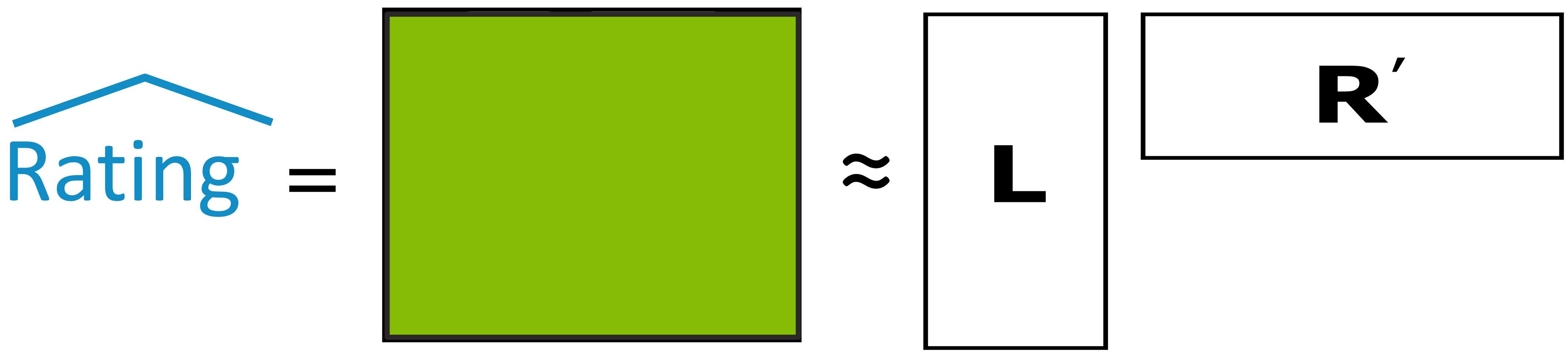
Movie ID	Feature vector
1	(3, 1)
2	(1, 2)
3	(2, 1)

Call this 3x2 matrix *R*

- Such prediction can be computed for all (user,movie) pairs
- And be written in a matrix form:

$$\begin{bmatrix} 6 & 2 & 4 \\ 4 & 3 & 3 \\ 1 & 2 & 1 \\ 7 & 4 & 5 \end{bmatrix} = \underset{L}{\begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 1 \\ 2 & 1 \end{bmatrix}} \underset{R^T}{\begin{bmatrix} 3 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix}}$$

Predictions in a matrix form



- Ratings matrix is the product of L and R: user feature matrix and movie feature matrix
- How do we learn the feature matrices from data?
- When we have all the ratings, then it is easy
  - PCA gives optimal factorization ***L*** and ***R*** in terms of reconstruction error
- This automatically discovers the right topics from data

- But, if we have all the ratings, we don't need to predict anything

Matrix factorizations are not unique

- Let's say we have an exact factorization ***M = L*R^T***

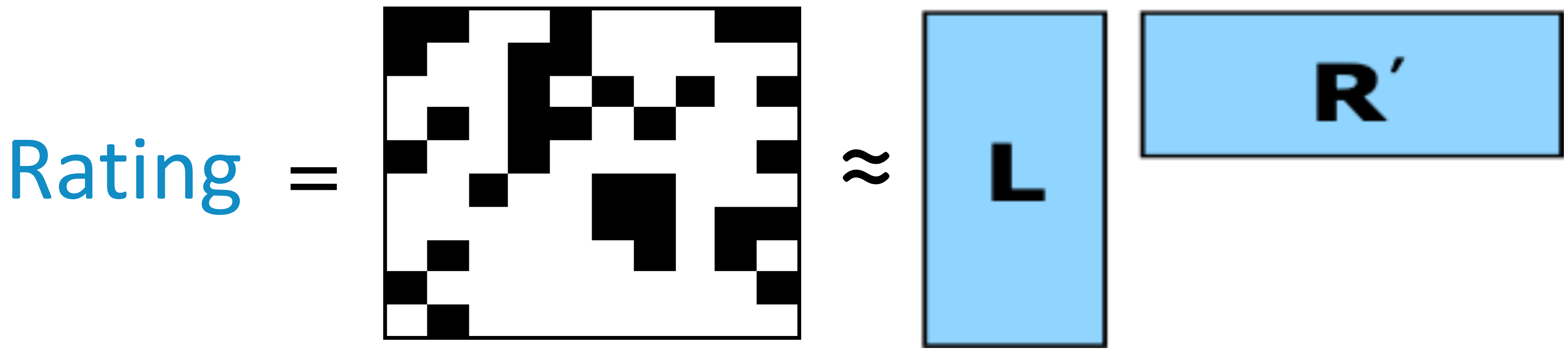
$$
\underset{M}{\begin{bmatrix} 6 & 2 & 4 \\ 4 & 3 & 3 \\ 1 & 2 & 1 \\ 7 & 4 & 5 \end{bmatrix}} = \underset{L}{\begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 1 \\ 2 & 1 \end{bmatrix}} \underset{R^T}{\begin{bmatrix} 3 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix}}
$$

- There are infinitely many factorizations, which give the exactly same ***M***
- For example, we can scale up the user features and scale down the movie ones, so that the ratings do not change

$$
\underset{M}{\begin{bmatrix} 6 & 2 & 4 \\ 4 & 3 & 3 \\ 1 & 2 & 1 \\ 7 & 4 & 5 \end{bmatrix}} = \underset{L}{\begin{bmatrix} 4 & 0 \\ 2 & 2 \\ 0 & 2 \\ 4 & 2 \end{bmatrix}} \underset{R^T}{\begin{bmatrix} 1.5 & 0.5 & 1.0 \\ 0.5 & 1.0 & 0.5 \end{bmatrix}}
$$

- Precisely, for any invertible matrix ***Q***, (LQ, RQ^{-T}) give the same matrix as (L,R) since $LQ*(RQ^{-T})^T = LQ*Q^{-T}R = LR = M$

From factorization to Matrix completion



- In reality, we only have partial observations of the ratings matrix
- We **fit** the best ***L*** and ***R***, to the observed ratings
- There has been many efficient algorithms to find factorization based on partial observations, a.k.a. **matrix completion problem**

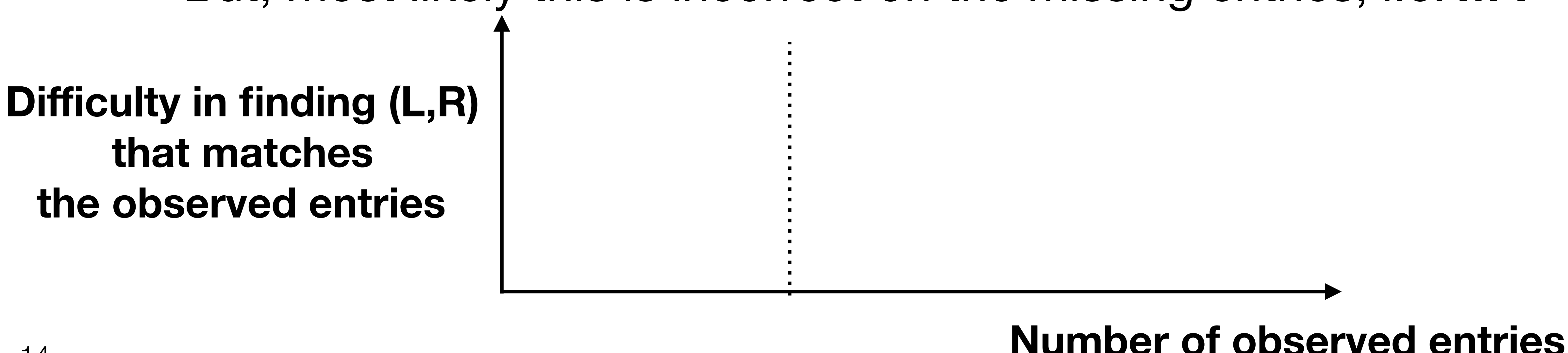
- We suppose there are ***m*** movies and ***n*** users, and ***k*** topics, and the ground truth matrix M is generated by $M=L_0*R_0^T$ for the form above

- No matter how many entries I observed, there are multiple choices of parameters ***(L,R)*** that will match all the entries
  - because, if ***(L,R)*** matches the entries, so does (LQ,RQ^{-T})

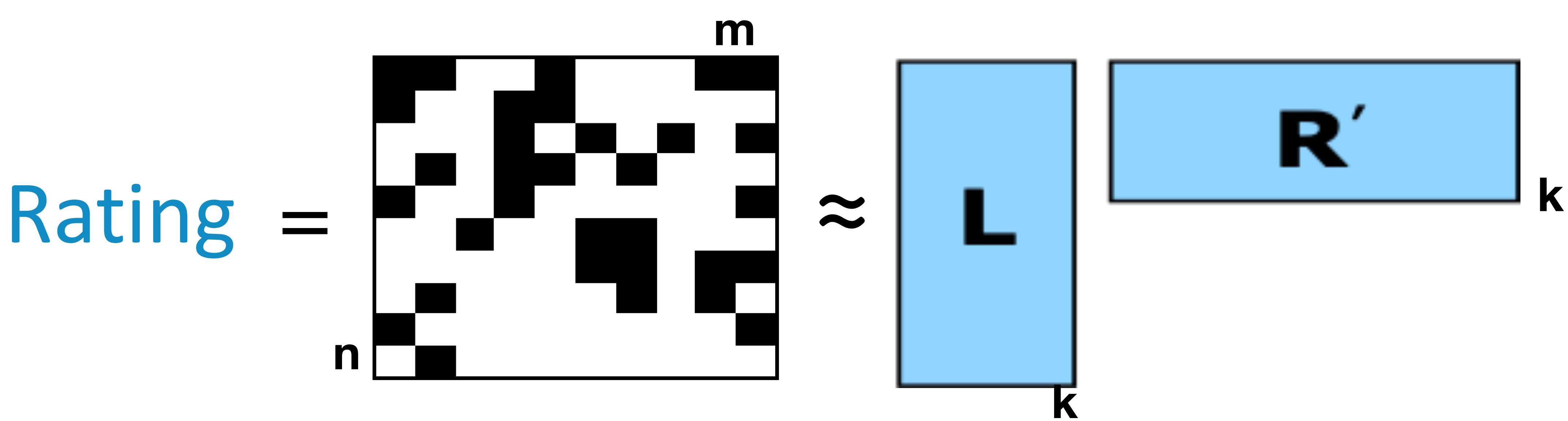
$$\begin{bmatrix} 6 & 2 & 4 \\ 4 & 3 & 3 \\ 1 & 2 & 1 \\ 7 & 4 & 5 \end{bmatrix} = \begin{bmatrix} & \\ & \\ & \\ & \end{bmatrix} \begin{bmatrix} & & \\ & & \end{bmatrix} \qquad \begin{bmatrix} & & \\ & & \\ & 2 & \\ & & \end{bmatrix} = \begin{bmatrix} & \\ & \\ & \\ & \end{bmatrix} \begin{bmatrix} & & \\ & & \end{bmatrix}$$

- But when can we solve this problem?
- That is how many entries do we need to see, in order for our prediction to be accurate?

- One extreme: suppose we observe all entries, then
  - Any factorization methods like singular value decomposition (SVD) will provide (one of the) correct factorizations
  - And, this correct, i.e. resulting $\boldsymbol{M = LR^T}$
- Another extreme: suppose we observe one entry, then
  - It is easy to match the entry observed
  - But, most likely this is incorrect on the missing entries, i.e. $\boldsymbol{M \text{ != } LR^T}$

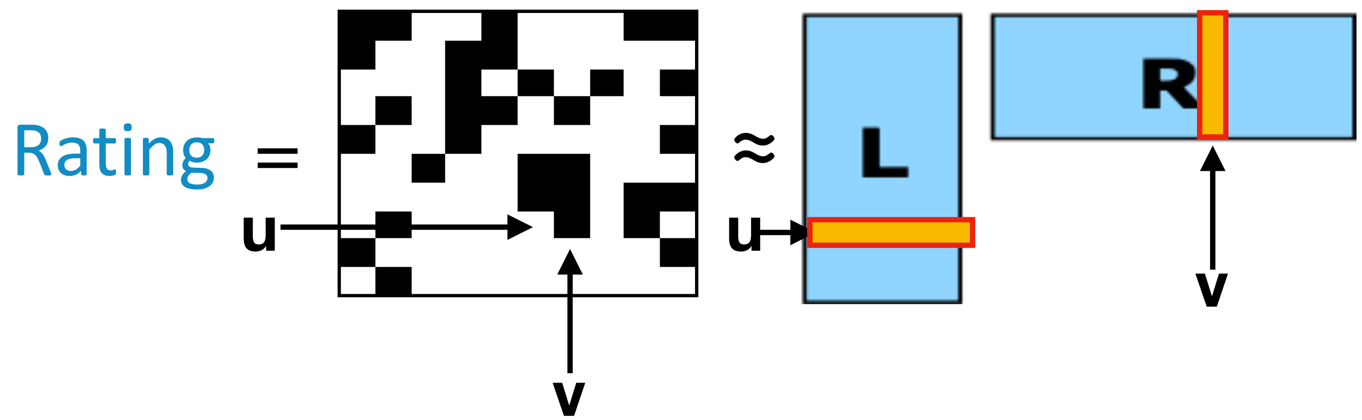


From factorization to Matrix completion



- If there are ***m*** movies and ***n*** users, and ***k*** topics, then how many parameters do we have in our factorization model ***L*** and ***R***?
  degrees-of-freedom = ***k*m+k*n***
- This is also sometimes called the degrees-of-freedom in the problem.
- How many entries do we observe if we have the full matrix?
- How many entries do you think we need, to accurately reconstruct the ground truth ***L*** and ***R*** that generated the data?

Algorithmic solution for matrix completion



- How do we write a program to find ***(L,R)*** matching the observed entries?
- Machine learning approach:
  - Write a loss function and minimize

$$\text{minimize}_{L,R} \sum_{u,v:r_{uv}\neq ?} \Big(\underbrace{(LR^T)_{uv}}_{L_u^T R_v} - r_{uv} \Big)^2$$

  - Coordinate descent us popular in solving this optimization

Coordinate descent

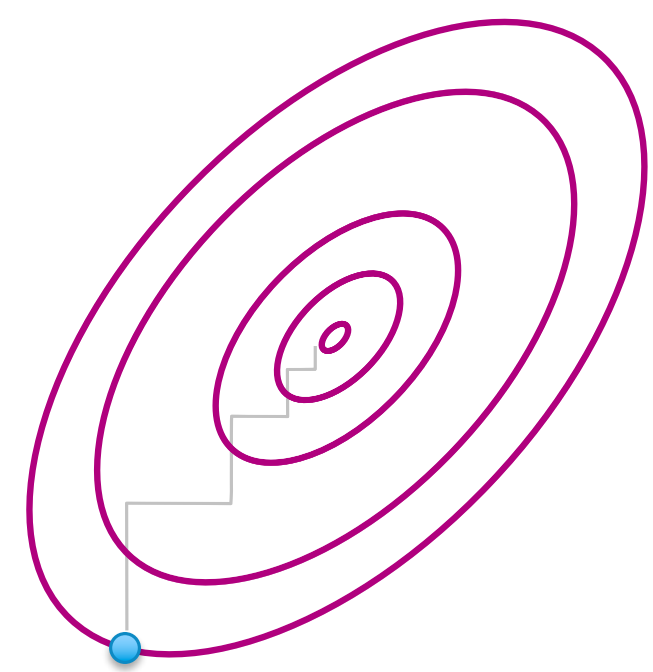
- Consider a optimization problem (in 2-dimensions for illustration purposes)

$$\text{minimize}_{w_0, w_1} \quad g(w_0, w_1)$$

- One method is called **coordinate descent**

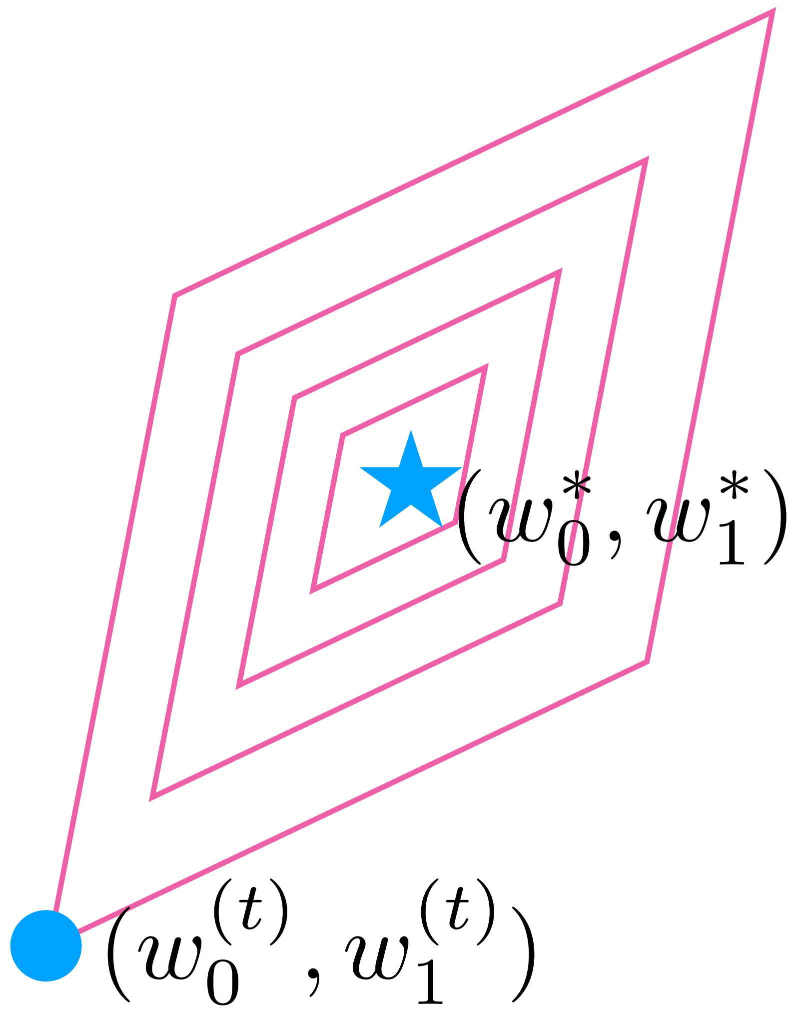
- Initialize ***(w_0,w_1)*** to be random or smart initialization
- While not converged, repeat
  - Pick a coordinate j in {0,1} (either random, round-robin, etc.)

$$w_j \leftarrow \arg\min_{w_j} \quad g(w_0, w_1)$$



- Main idea:
  - Minimizing over 1 coordinate is much easier
  - No need to choose step-size
  - This is guaranteed to find optimal solution, under some constraints
- When does it fail?

Coordinate descent



- Coordinate descent successfully finds the optimal solution if g(.) is **strongly convex** and **smooth**

Coordinate descent for matrix completion

$$\text{minimize}_{L,R} \sum_{u,v:r_{uv}\neq ?} \Big(\underbrace{(LR^T)_{uv}}_{L_u^T R_v} - r_{uv} \Big)^2$$

- Initialize ***(L,R)***
- Repeat
  - Fix ***R*** and optimize over ***L***
  - Fix ***L*** and optimize over ***R***

  - First insight:

$$\min_{L_1,\cdots,L_n} \sum_{(u,v):r_{uv}\neq ?} (L_u^T R_v - r_{uv})^2$$

$$= \min_{L_1,\cdots,L_n} \sum_{u=1}^{n} \Big\{ \sum_{v:r_{uv}\neq ?} (L_u^T R_v - r_{uv})^2 \Big\}$$

$$= \sum_{u=1}^{n} \Big\{ \min_{L_u} \sum_{v:r_{uv}\neq ?} (L_u^T R_v - r_{uv})^2 \Big\}$$

This only involves each row of L independently, and can be solved as a separate optimization for each row

Coordinate descent for matrix completion

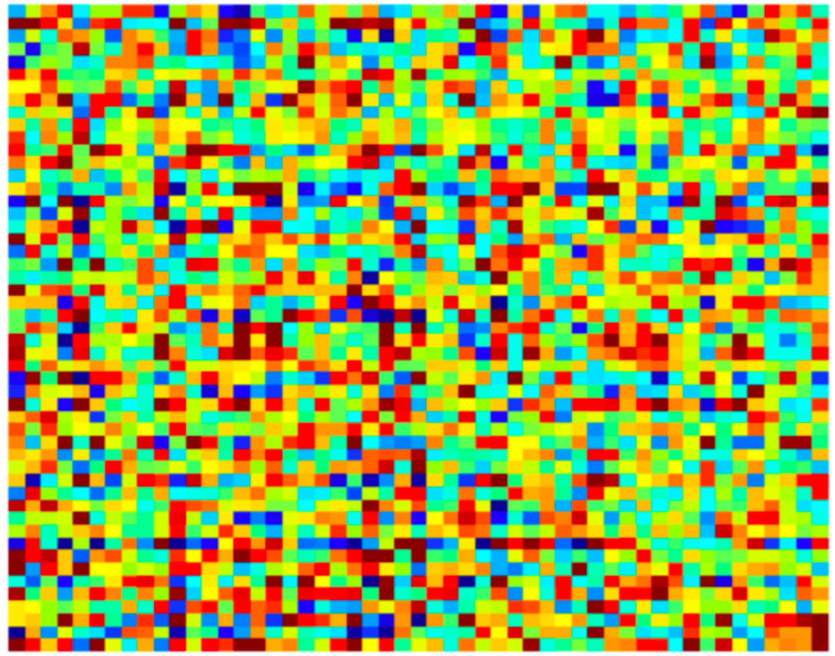
- We broke down the problem into solving multiple inner optimizations of the form:

$$\min_{L_u} \sum_{v: r_{uv} \neq ?} (L_u^T R_v - r_{uv})^2$$

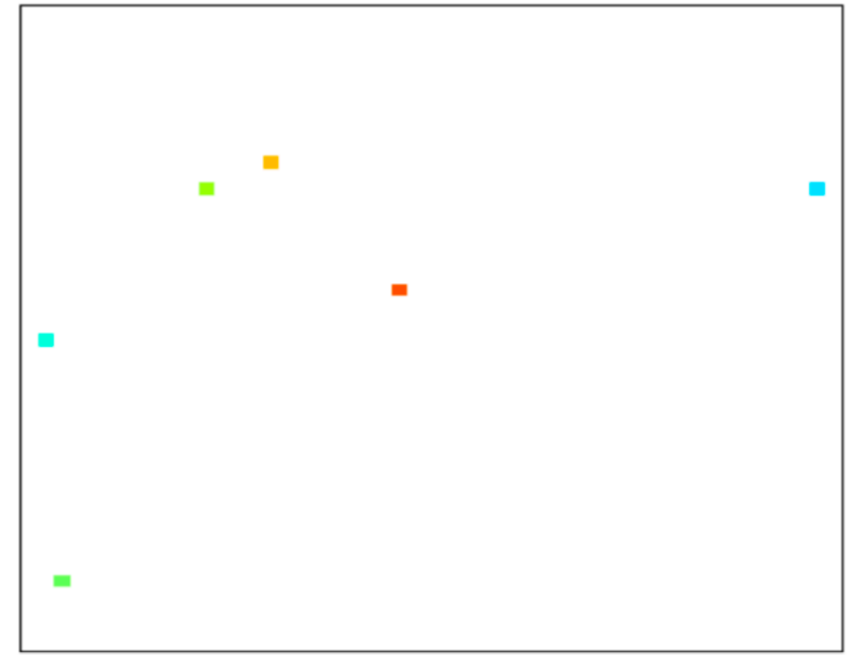
- Second insight:
  - And this is the standard linear regression with quadratic loss
  - Many efficient solvers exist + can be solved in a closed form too
-

Example: 2000×2000 rank-8 random matrix

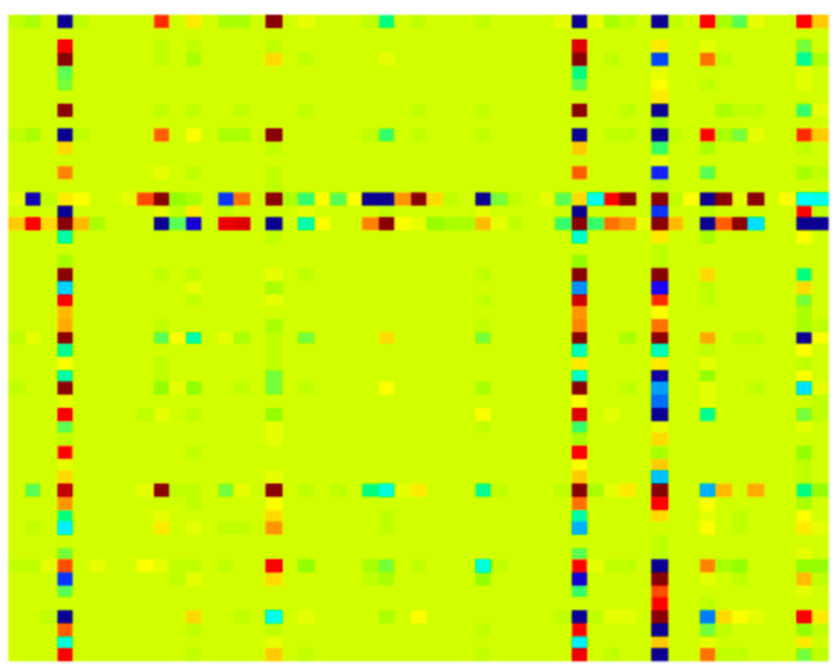
low-rank matrix M



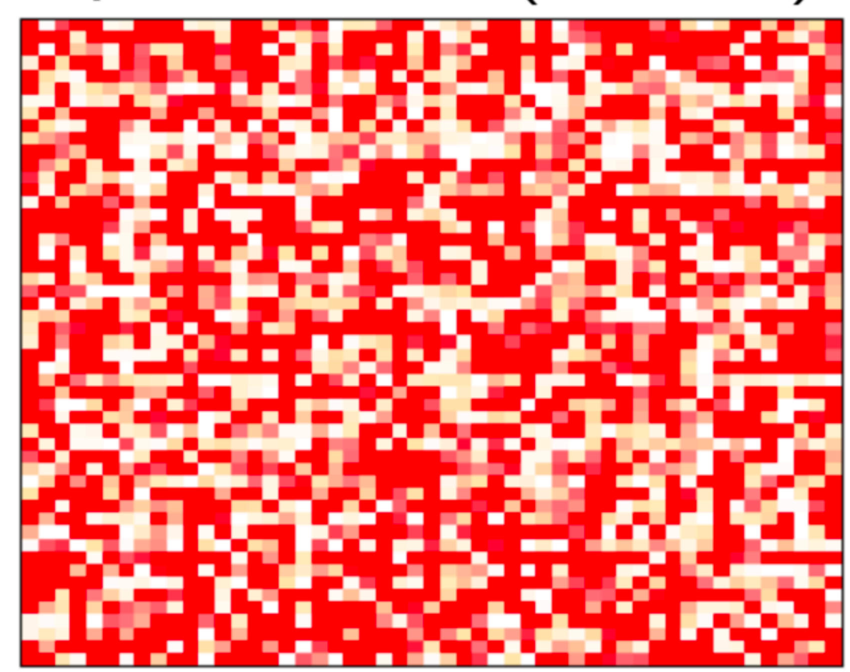
sampled matrix M^E



OptSpace output $\widehat{M}$



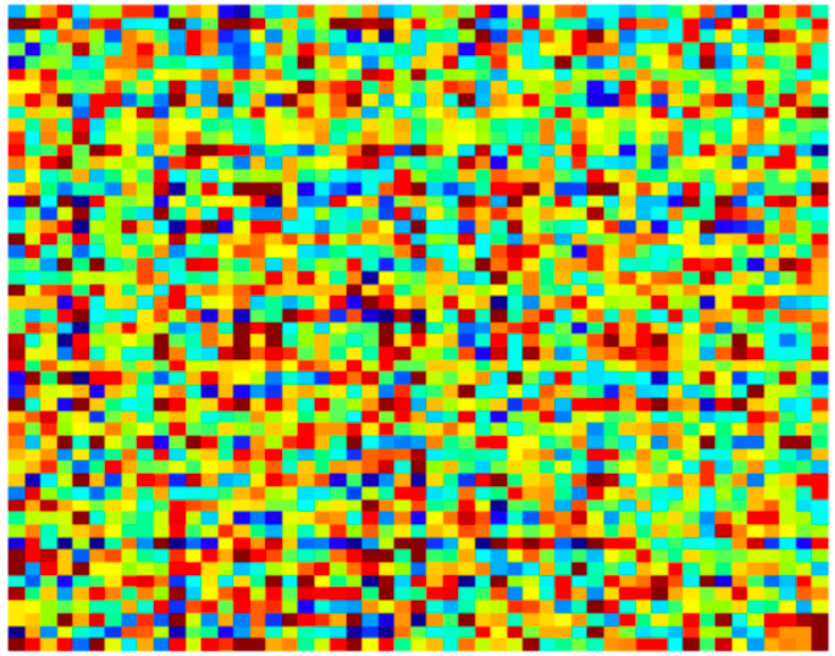
squared error $(M - \widehat{M})^2$



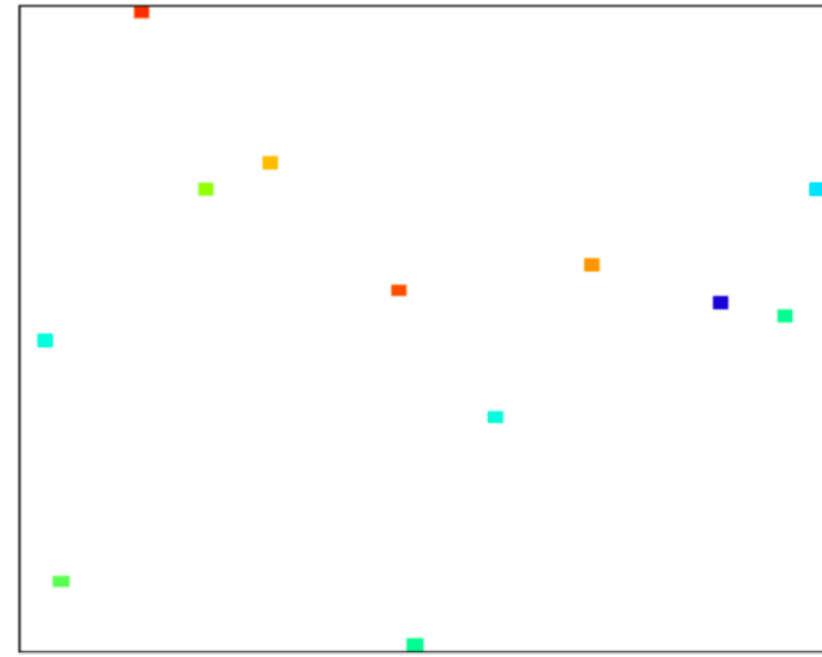
0.25% sampled

Example: 2000×2000 rank-8 random matrix

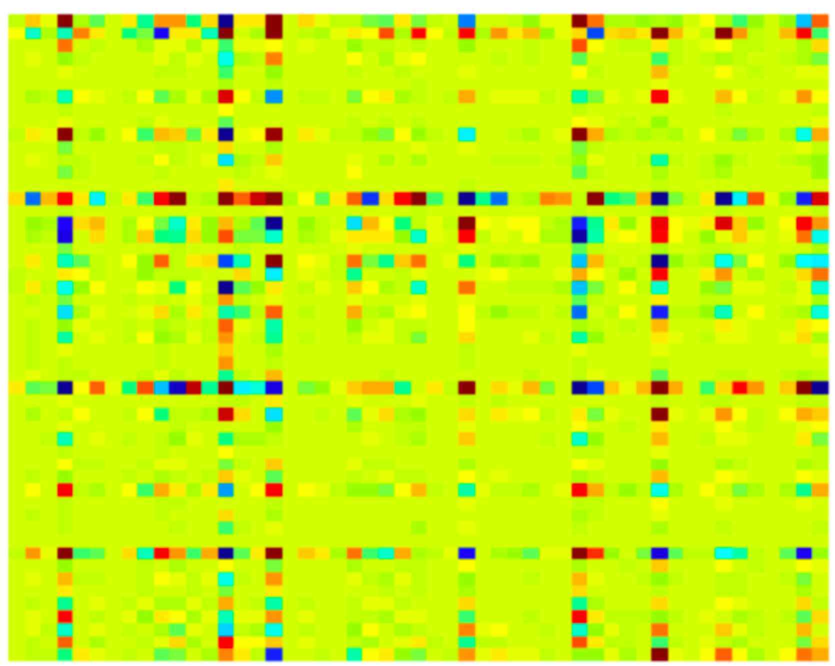
low-rank matrix M



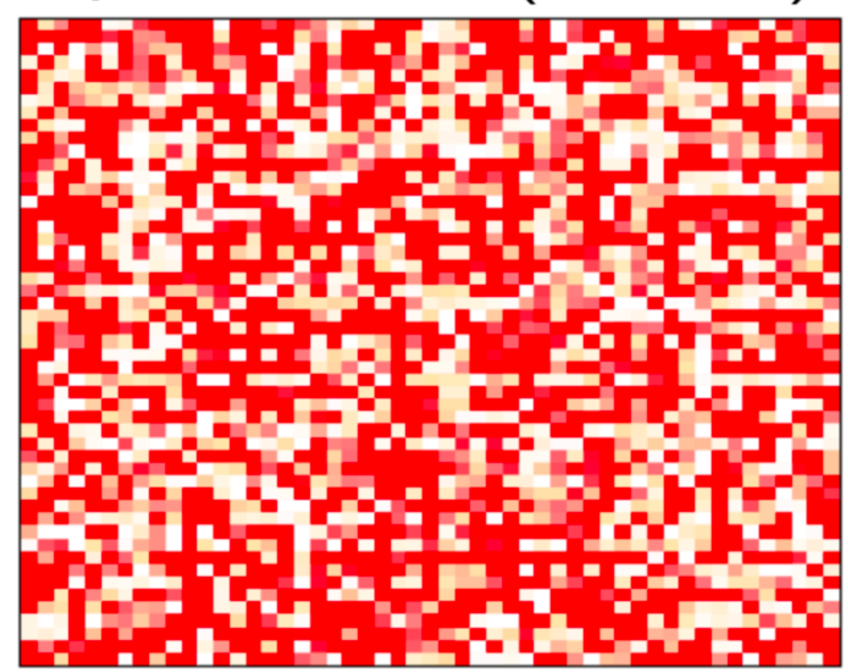
sampled matrix M^E



OptSpace output $\widehat{M}$

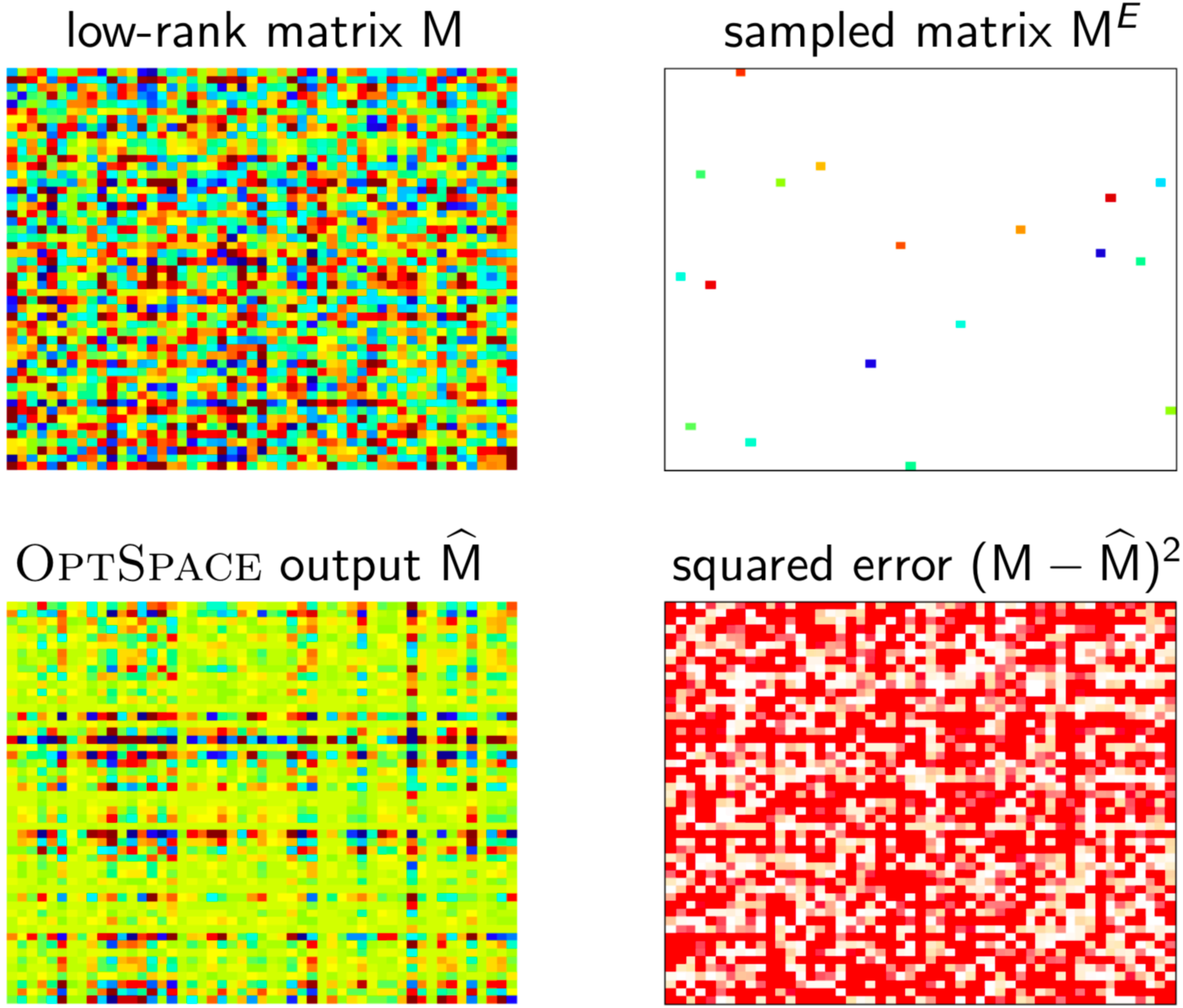


squared error $(M - \widehat{M})^2$



0.50% sampled

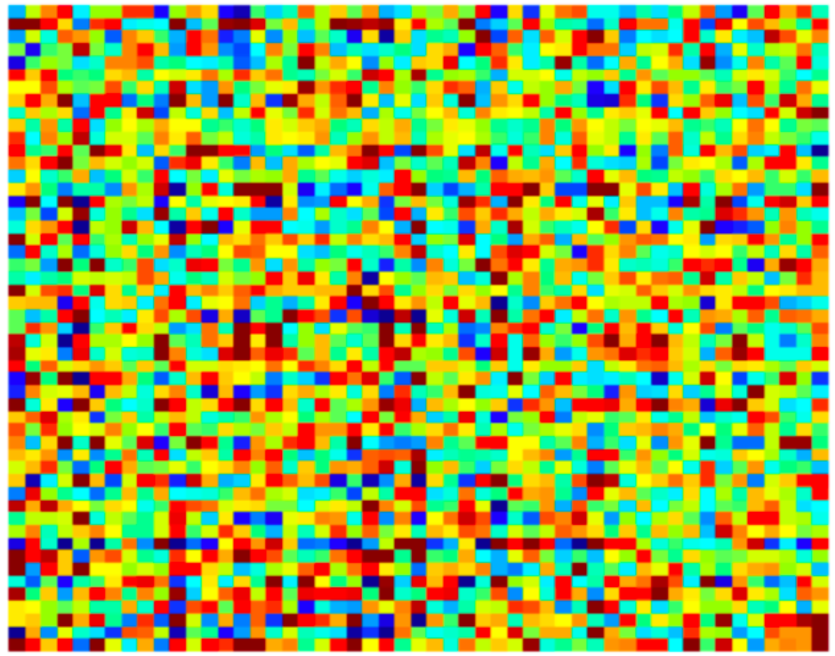
Example: 2000×2000 rank-8 random matrix



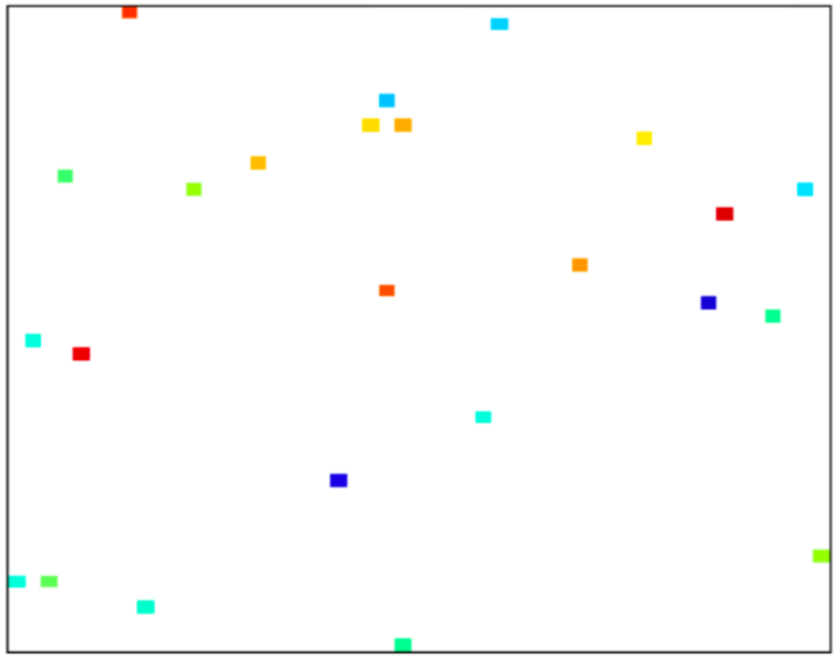
0.75% sampled

Example: 2000 $\times$ 2000 rank-8 random matrix

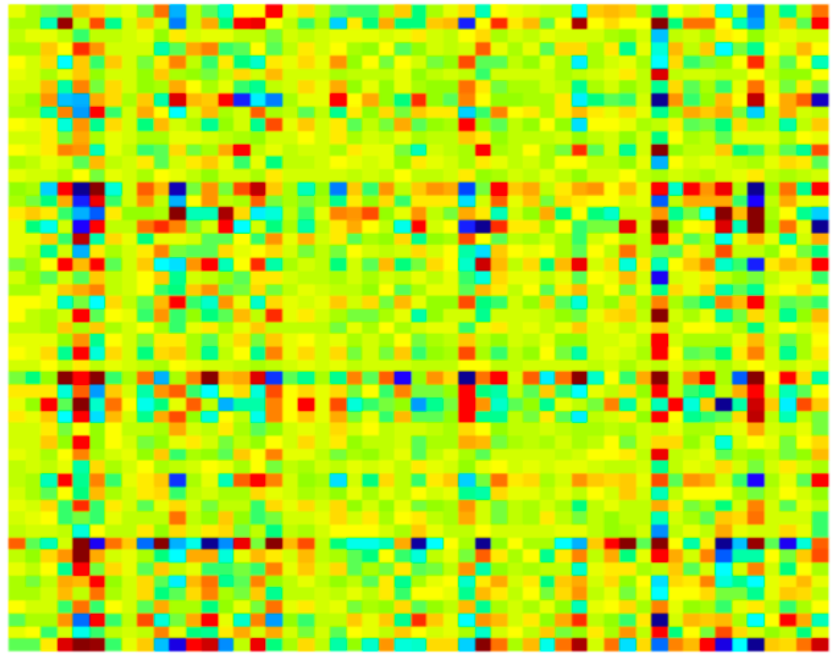
low-rank matrix M



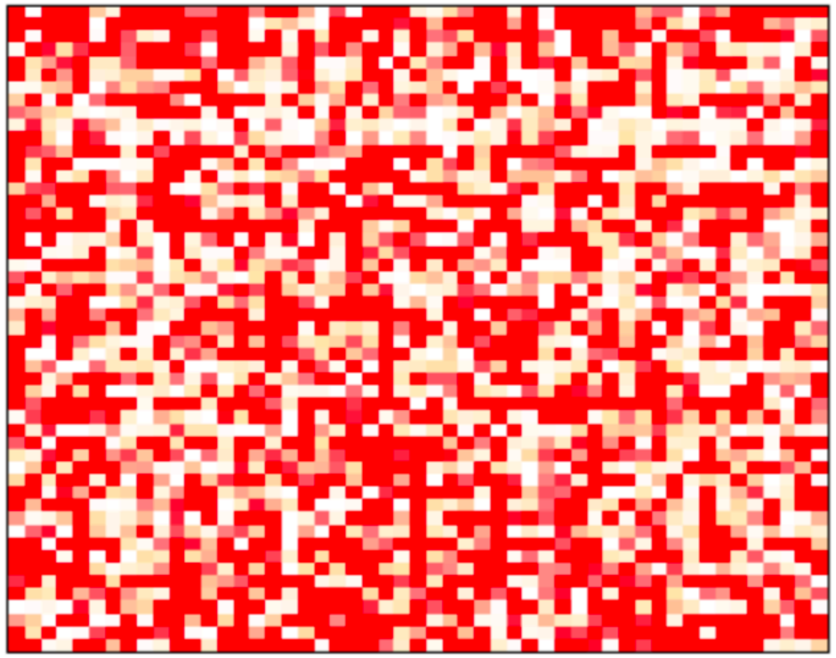
sampled matrix M^E



OptSpace output $\widehat{M}$



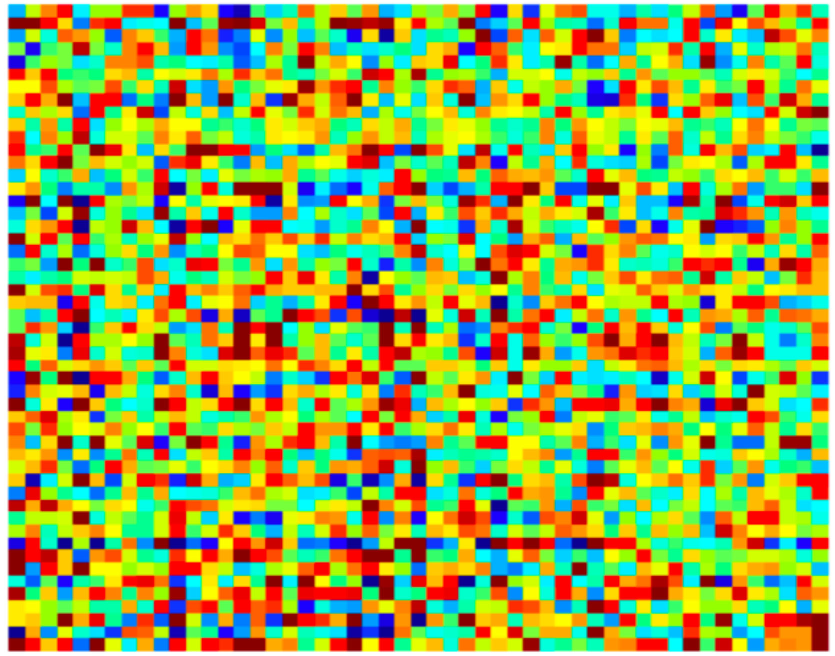
squared error $(M - \widehat{M})^2$



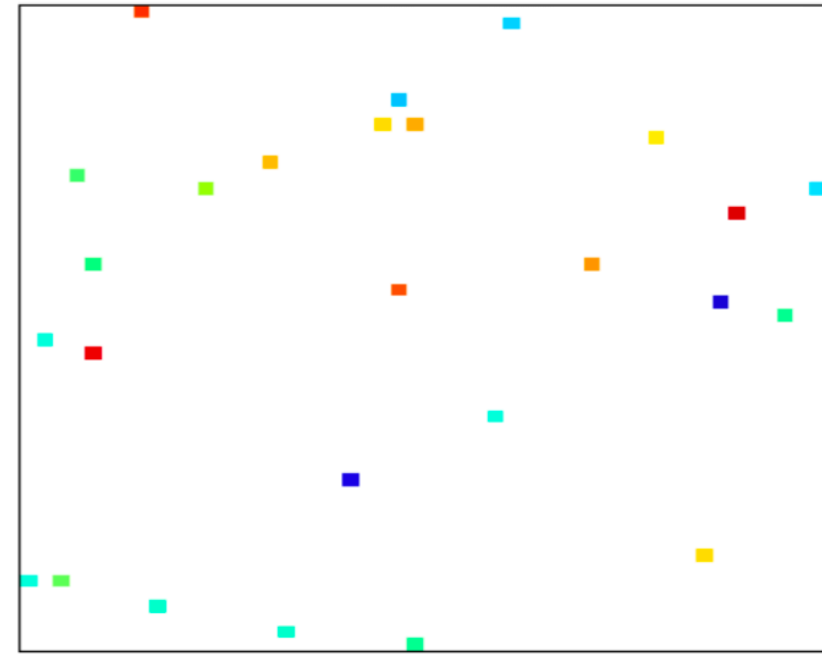
1.00% sampled

Example: 2000×2000 rank-8 random matrix

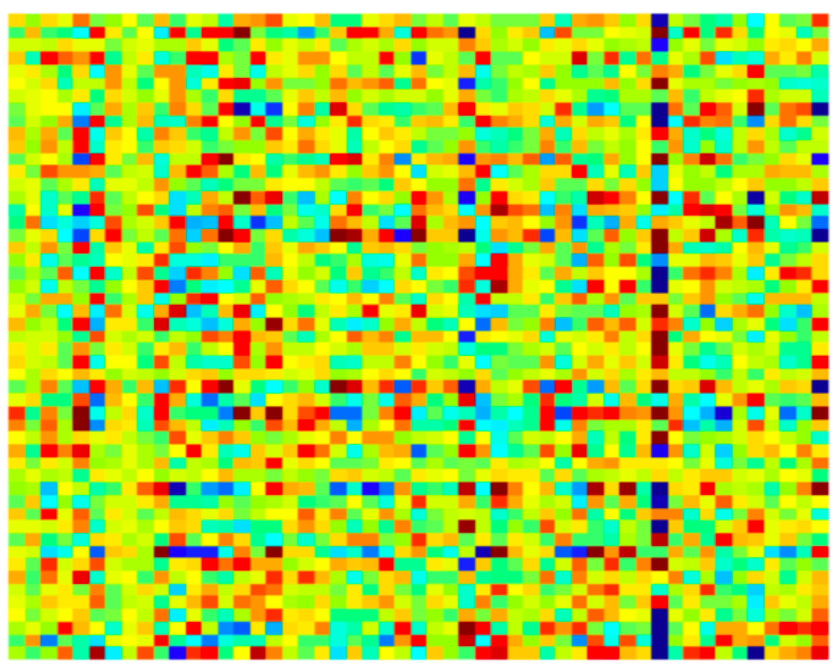
low-rank matrix M



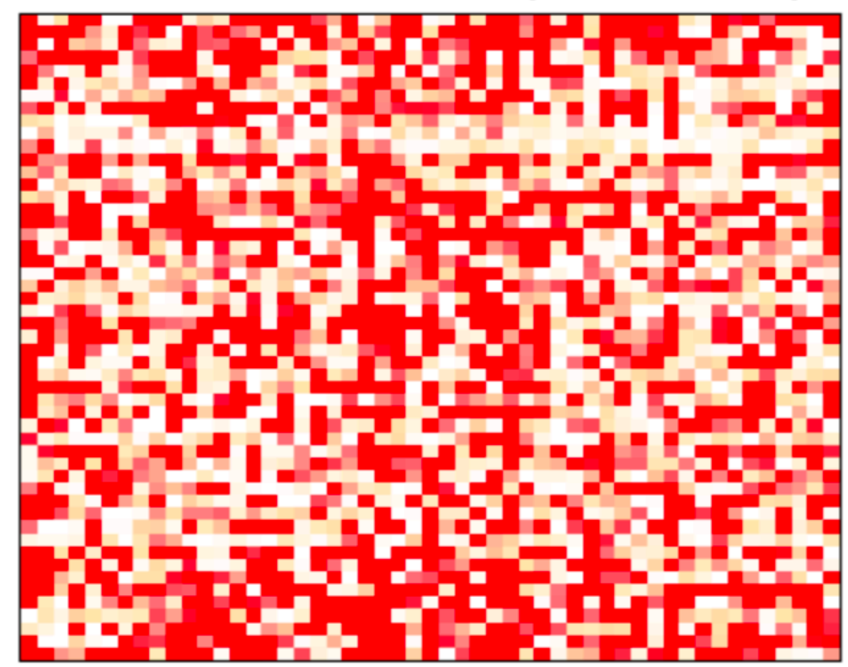
sampled matrix M^E



OptSpace output $\widehat{M}$



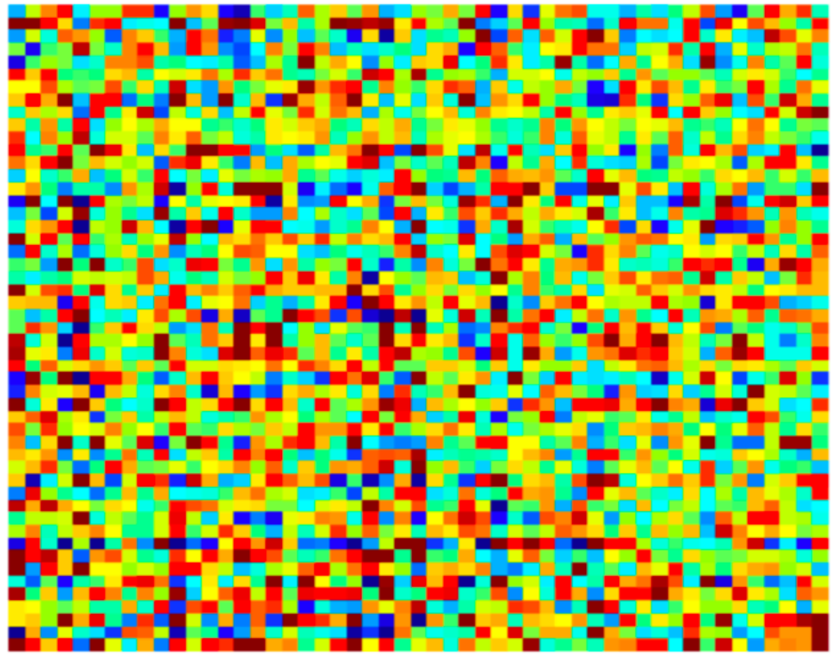
squared error $(M - \widehat{M})^2$



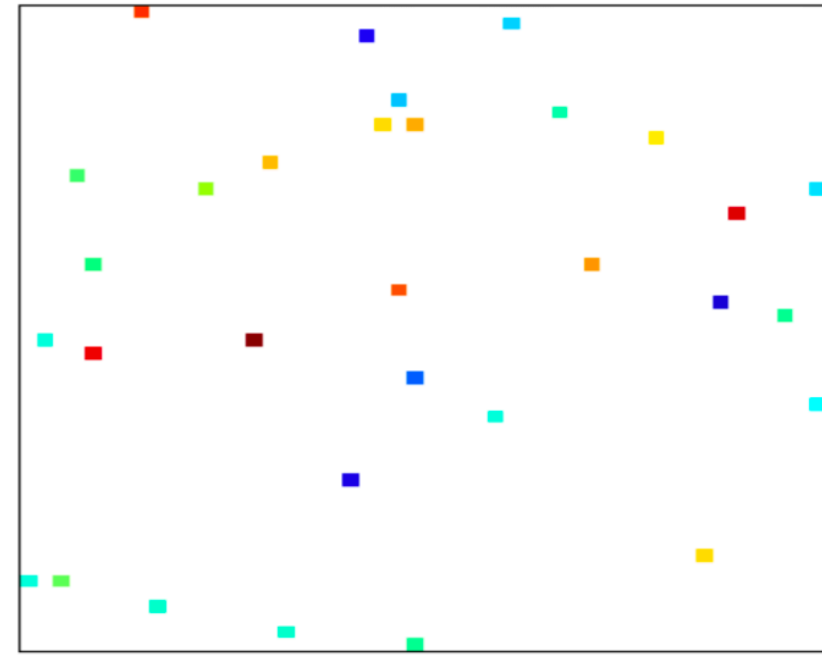
1.25% sampled

Example: 2000×2000 rank-8 random matrix

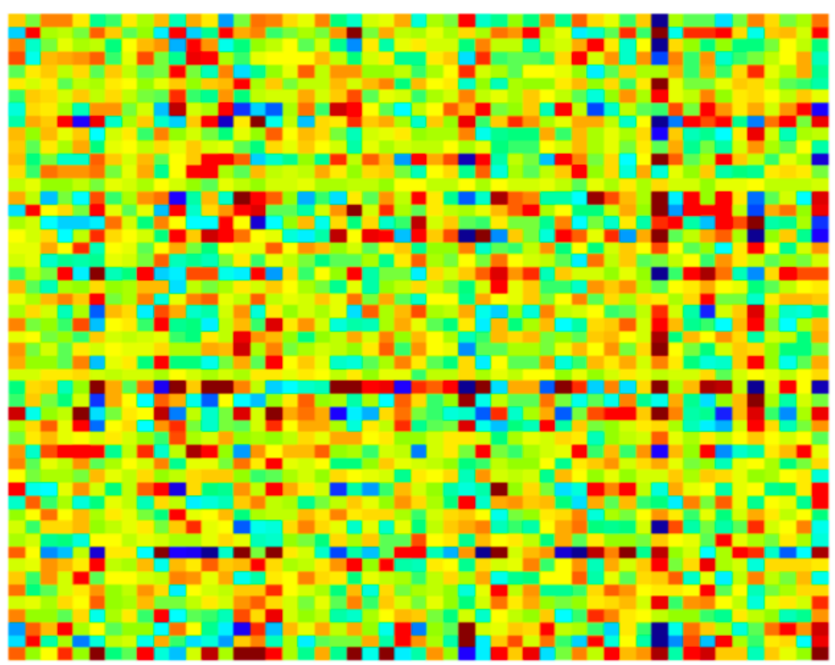
low-rank matrix M



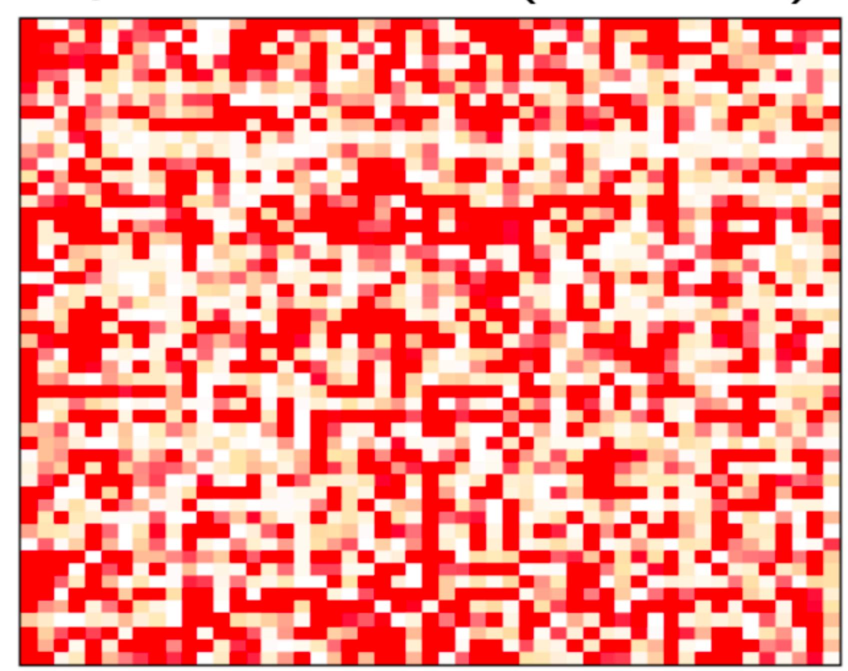
sampled matrix M^E



OptSpace output $\widehat{M}$



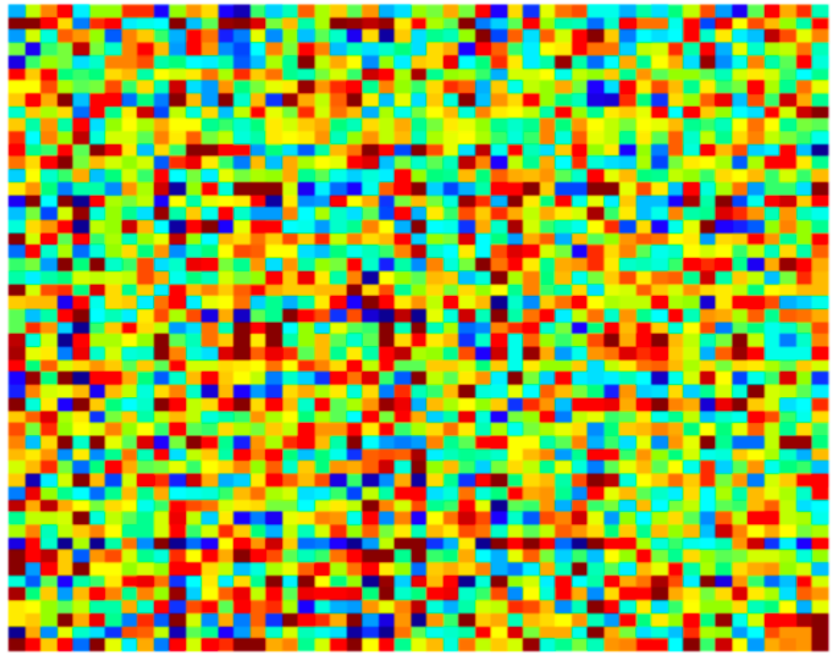
squared error $(M - \widehat{M})^2$



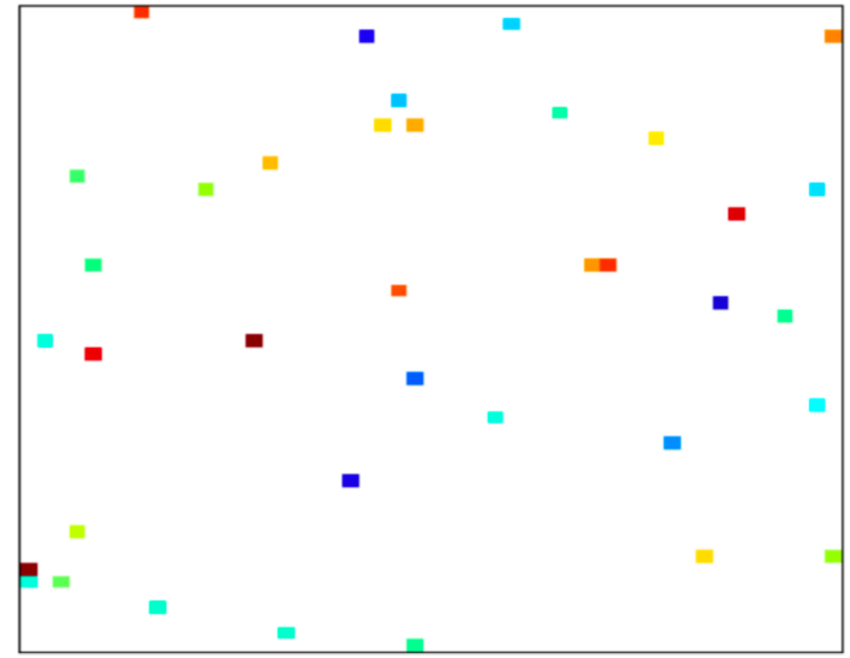
1.50% sampled

Example: 2000×2000 rank-8 random matrix

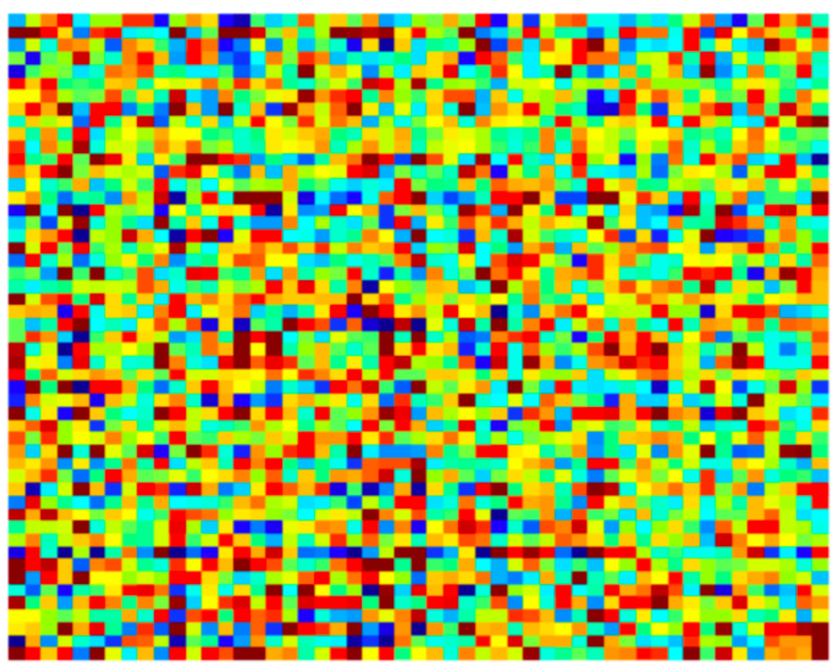
low-rank matrix M



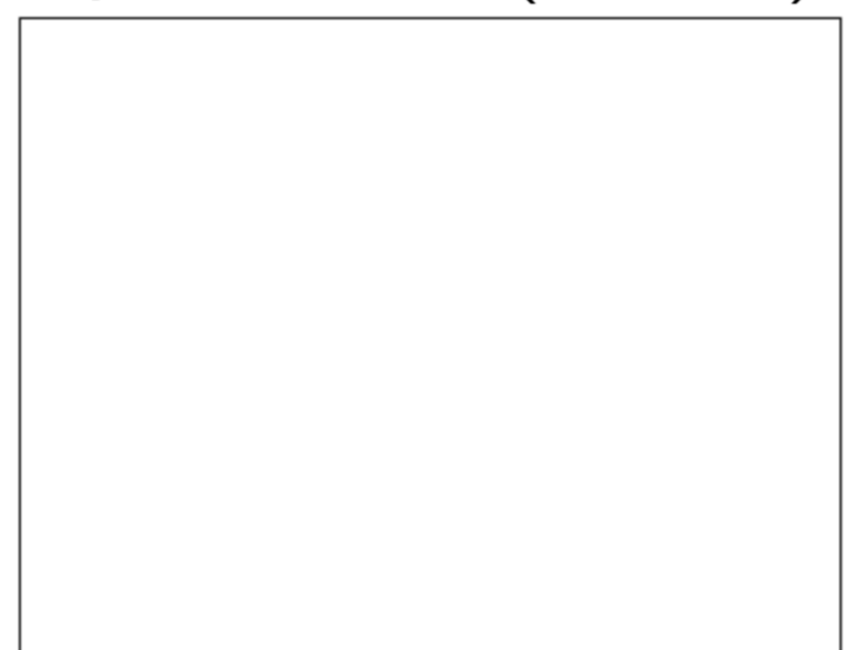
sampled matrix M^E



OptSpace output $\widehat{M}$

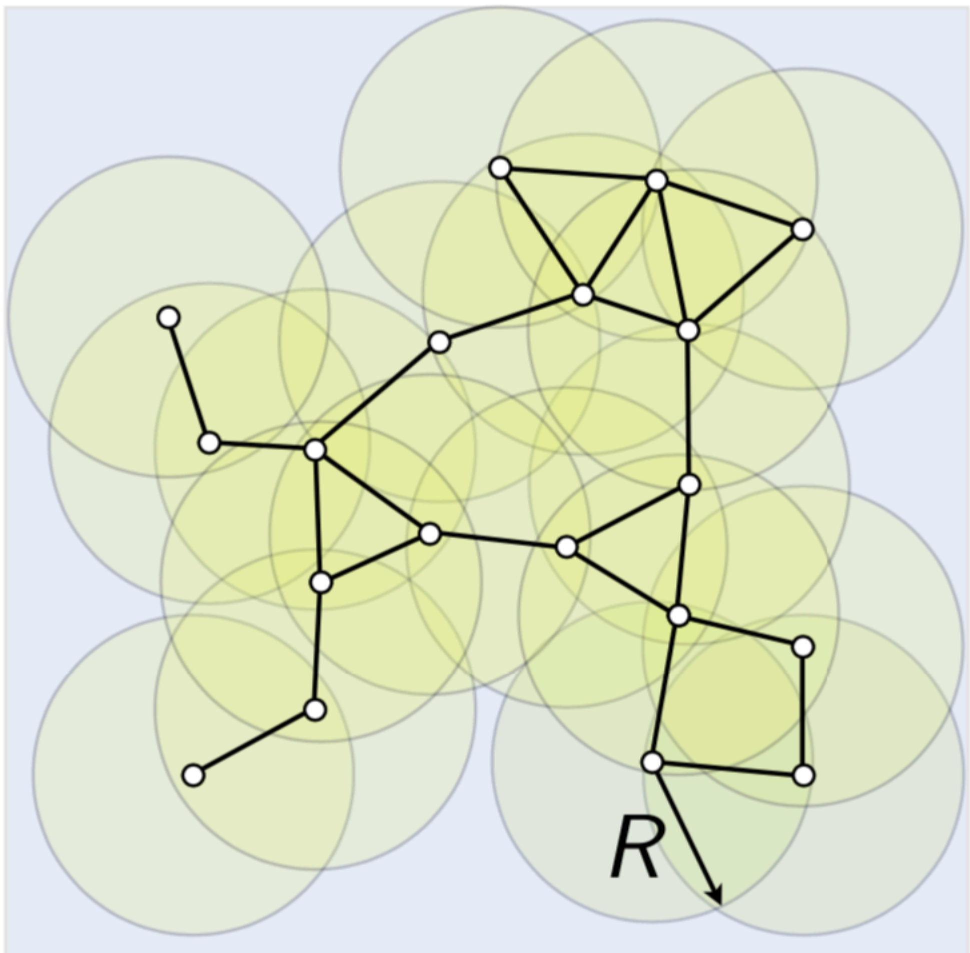


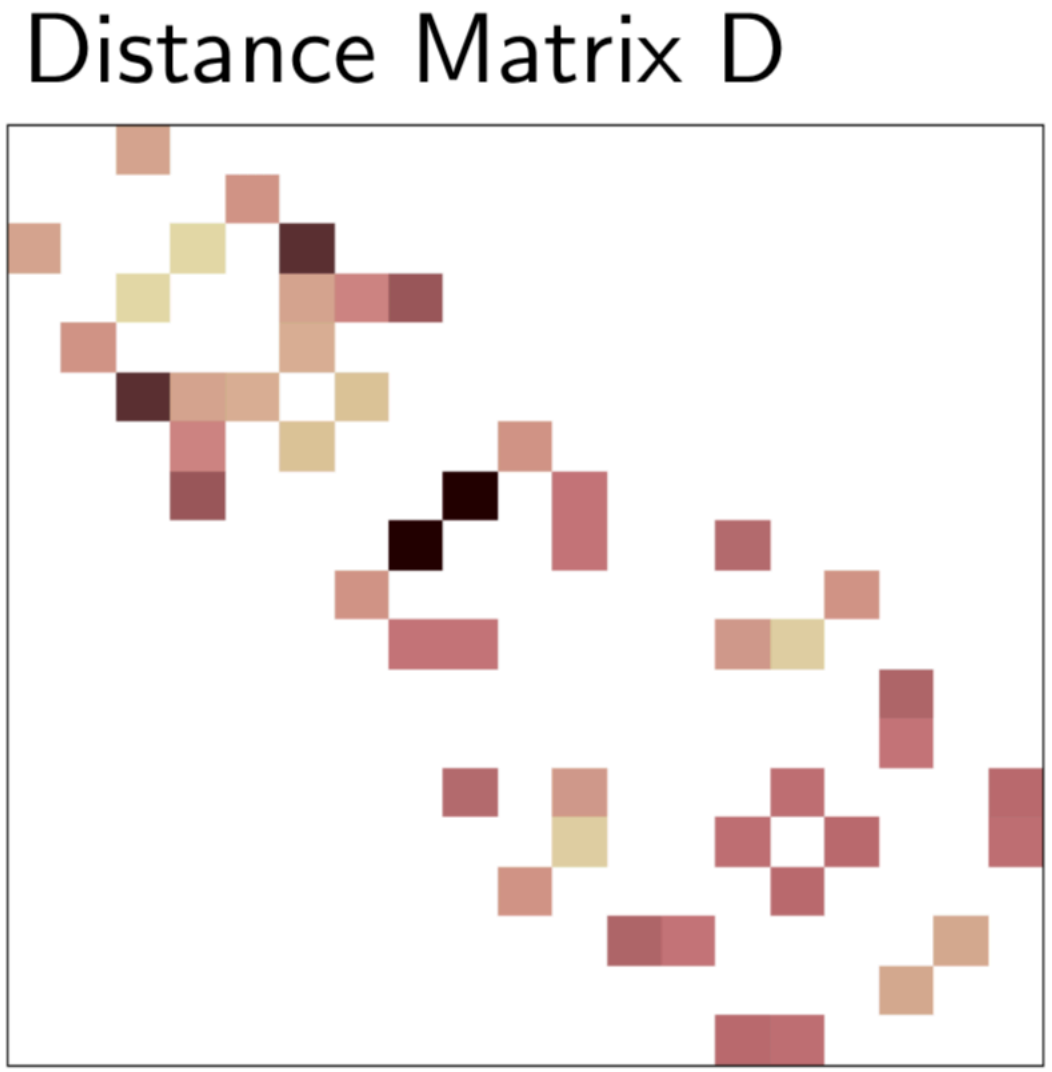
squared error $(M - \widehat{M})^2$



1.75% sampled

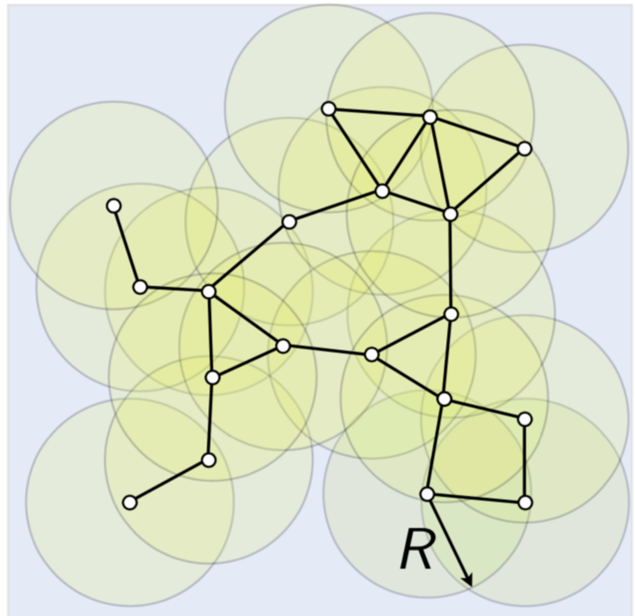
Application: localization



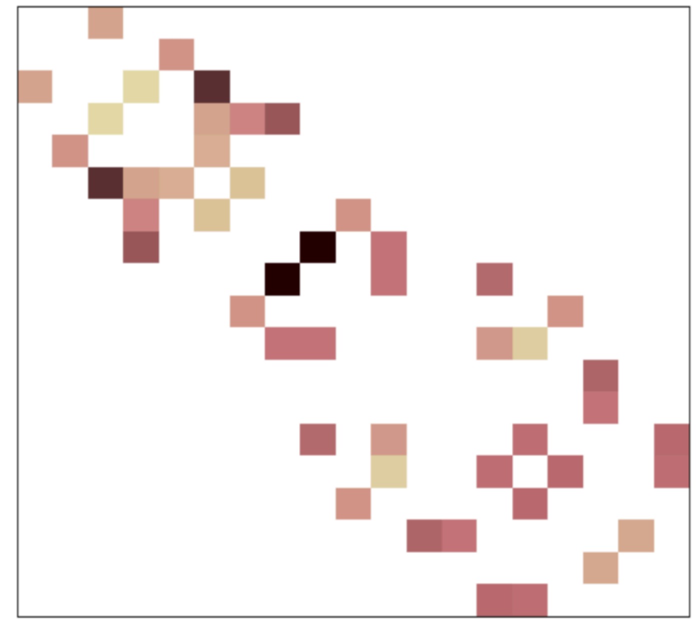


- Wireless sensors deployed in a region
- Each measure distance to the close-by sensors
- Goal: find the distances to all sensors
  - If we have all pairwise distances, then it is easy to find locations of all sensors simultaneously

Application: localization



Distance Matrix D



- Why is this a **Matrix Completion problem**?
  - We have missing entries
  - The data is in a matrix form
  - But most importantly, the ground-truth is a low-rank matrix
    - The ambient dimension is 2 or 3, i.e. position is (x_u, y_u)

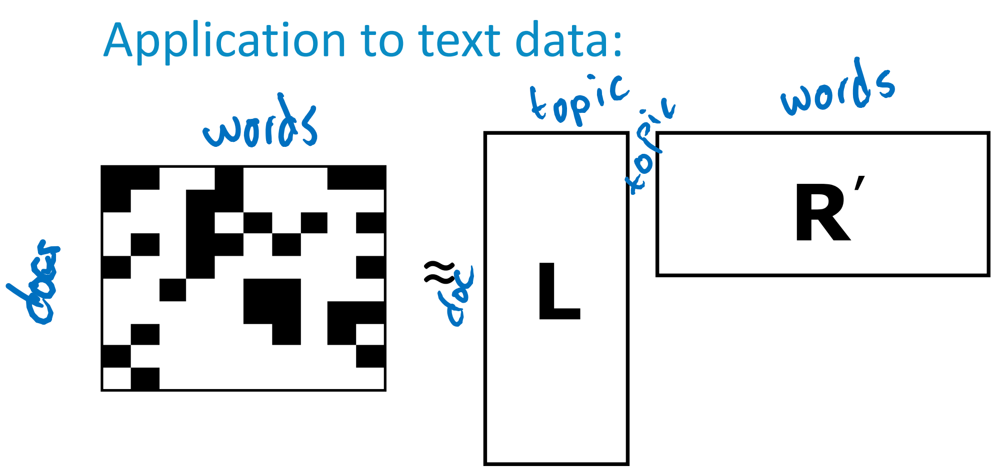
$$D_{uv} = (x_u - x_v)^2 + (y_u - y_v)^2$$

$$D = \overset{L}{\begin{bmatrix} 1 & x_u^2+y_u^2 & -\sqrt{2}x_u & -\sqrt{2}y_u \\ & & & \\ & & & \\ & & & \end{bmatrix}} \overset{R^T}{\begin{bmatrix} x_v^2+y_v^2 & & \\ 1 & & \\ \sqrt{2}x_v & & \\ \sqrt{2}y_v & & \end{bmatrix}}$$

Application: recommendation systems

- Given partially observed ratings matrix
- Discover ***k*** topics, and ***k***-dimensional user features $\boldsymbol{L_u}$ movie features $\boldsymbol{R_v}$
- Predict how much a user will like a movie by $\boldsymbol{r_{uv}=L_u^T * R_v}$
- Make recommendations based on the prediction

- Applied to Wikipedia



Which is correct about matrix factorization based recommendation systems?

- a) provide personalization
- b) capture context (e.g. time of the day)

- Another weakness of matrix factorization
  - We need to know ***k***, in some sense
  - If we set k= min{m,n}, what goes wrong?
    - overfitting

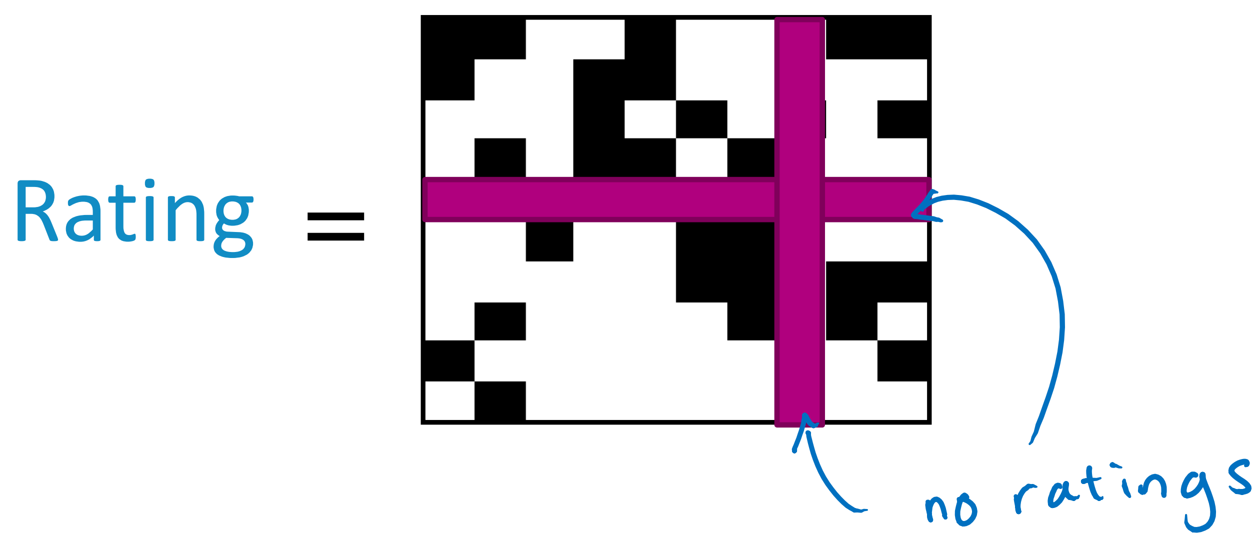
$$\begin{bmatrix} 6 & & 4 \\ 4 & 3 & \\ & 2 & \\ 7 & & 5 \end{bmatrix} = \underbrace{\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}}_{R^T}$$

  - Solution: regularize

$$\min_{L_u} \sum_{v: r_{uv} \neq ?} (L_u^T R_v - r_{uv})^2 + \lambda \|L_u\|^2$$

Featured matrix factorization

- Limitations of matrix factorization
  - Cold-start problem
    - This model still cannot handle a new user or movie



- As there is no observation for the entire row/column putting anything in that row has no penalty

$$\text{minimize}_{L,R} \sum_{u,v:r_{uv}\neq ?} \Big(\underbrace{(LR^T)_{uv}}_{L_u^T R_v} - r_{uv} \Big)^2$$

Combining features and discovered topics

- Features capture context
  - *Time of day, what I just saw, user info, past purchases,…*

- Discovered topics from matrix factorization capture groups of users who behave similarly
  - *Women from Seattle who teach and have a baby*

- Combine to mitigate cold-start problem
  - Ratings for a new user from features only
  - As more information about user is discovered, matrix factorization topics become more relevant

Collaborative filtering with specified features

- Create feature vector for each movie (often have this even for new movies):

genre, year, director, . . .

$\phi(v) =$ ('action', 1994, Tarantino, ...)

- Define weights on these features for how much **all users** like each feature

$w =$ vector of same length

- Fit linear model:

For all users, $r_{uv} \approx w \cdot \phi(v)$ standard regression model

- Minimize:

$$\min_{w} \sum_{r_{uv} \neq ?} (w \cdot \phi(v) - r_{uv})^2 + \lambda_w \|w\| \leftarrow$$ LS, lasso, ridge

Building in personalization

- Of course, users do *not* have identical preferences
- Include a user-specific deviation from the global set of user weights:

$$r_{uv} \approx (w + w_u) \cdot \phi(v)$$

personalization to user u = deviation from crowd vector w

- If we don't have any observations about a user, use wisdom of the crowd

Initialize $w_u = 0 \Rightarrow r_{uv} \approx w \cdot \phi(v)$

- As we gain more information about the user, forget the crowd

w_u more informed (personalization)

- Can add in user-specific features, and cross-features, too

$\phi(u) = (25, F, MSc, \ldots)$ — age, gender, education

$\phi(u,v) = (\ldots \phi(u) \ldots, \ldots \phi(v) \ldots, \text{cross features})$

Featurized matrix factorization— A combined approach

Feature-based approach:

- Feature representation of user and movies fixed
- Can address cold-start problem

Matrix factorization approach:

- Suffers from cold-start problem
- User & movie features are learned from data

A unified model:

$r_{uv} \approx L_u \cdot R_v + (w + w_u) \cdot \phi(u,v)$

solve via coord. desc., grad. desc., etc.