Lecture 14:
Principal Component Analysis

- reduce the dimension of the data to visualize and understand

- Homework 3, due Wednesday, November 19th midnight

CSE 446/546 2025 Autumn

- Supervised learning: $\{(x_i, y_i)\}_{i=1}^n$
  - Linear models (lectures 1-9)
    - Linear regression
    - Ridge regression
    - LASSO regression
    - Logistic regression
  - Non-linear models (lectures 10-13)
    - Kernel methods
    - Neural Networks
    - Non-parametric methods
- **Unsupervised learning (lectures 14-16):** $\{x_i\}_{i=1}^n$
  - **PCA/SVD**
  - Clustering
- Modern machine learning (lectures 17-18)

Motivation: dimensionality reduction

- Definition:
  - **Dimensionality reduction**: Given $\{x_i\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$, represent each data point in a lower dimension $k < d$ in order to visualize data, understand the pattern, de-noise, and compress data.
- Key idea: many real data have patterns that repeat over samples
- Can we exploit this redundancy? Can we find some patterns and use them?
- Toy example: Food Ratings Data with $n = 4$ samples in $d = 4$ dimensions

	Alice=x_1	Bob=x_2	Carolyn=x_3	Dave=x_4
kale	10	7	2	3
taco bell	1	2	9	6
sushi	2	9	7	10
pop tarts	7	6	3	2

- This is **unsupervised**, in the sense that there is no label, just input data

Motivation: dimensionality reduction

- Toy example: Food Ratings Data x_1

	Alice=x_1	Bob=x_2	Carolyn=x_3	Dave=x_4
kale	10	7	2	3
taco bell	1	2	9	6
sushi	2	9	7	10
pop tarts	7	6	3	2

- Goal of **dimensionality reduction** is visualization, compression, and/or finding pattern.
- If we an represent each data point approximately as a combination of (the mean and) two **principal vectors** $u_1, u_2 \in \mathbb{R}^d$

$$x_i \approx \bar{x} + a_i[1] \cdot u_1 + a_i[2] \cdot u_2$$

with $\bar{x} = \begin{bmatrix} 5.5 \\ 4.5 \\ 5 \\ 5.5 \end{bmatrix}$, $u_1 = \begin{bmatrix} 3 \\ -3 \\ -3 \\ 3 \end{bmatrix}$, and $u_2 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$

Motivation: dimensionality reduction

	Alice=x_1	Bob=x_2	Carolyn=x_3	Dave=x_4
kale	10	7	2	3
taco bell	1	2	9	6
sushi	2	9	7	10
pop tarts	7	6	3	2

$$x_i \approx \bar{x} + a_i[1] \cdot u_1 + a_i[2] \cdot u_2$$

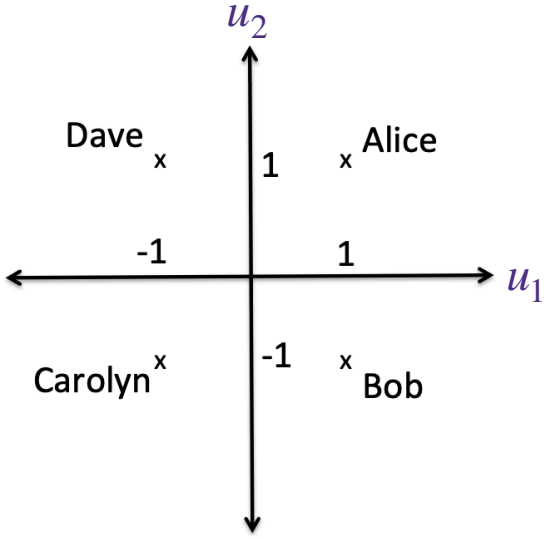
with $\bar{x} = \begin{bmatrix} 5.5 \\ 4.5 \\ 5 \\ 5.5 \end{bmatrix}$, $u_1 = \begin{bmatrix} 3 \\ -3 \\ -3 \\ 3 \end{bmatrix}$, and $u_2 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$. We can transform the data into

	Alice=x_1	Bob=x_2	Carolyn=x_3	Dave=x_4
$a_i[1]$	1	1	-1	-1
$a_i[2]$	1	-1	-1	1

What good is this new representation of the data?

Motivation: dimensionality reduction

	Alice=x_1	Bob=x_2	Carolyn=x_3	Dave=x_4
kale	10	7	2	3
taco bell	1	2	9	6
sushi	2	9	7	10
pop tarts	7	6	3	2
$a_i[1]$	1	1	-1	-1
$a_i[2]$	1	-1	-1	1



- **Visualization of data** can help understand the data and also group together data points with similar patterns (also known as clustering)
- **Interpretation of data** to understand the underlying pattern
  - What does u_1 mean?
  - What does u_2 mean?

$$\bar{x} = \begin{bmatrix} 5.5 \\ 4.5 \\ 5 \\ 5.5 \end{bmatrix}, \quad u_1 = \begin{bmatrix} 3 \\ -3 \\ -3 \\ 3 \end{bmatrix}, \quad \text{and} \quad u_2 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$$

- if normalized, u_1 and u_2 are called **principal components**

Goal of Principal Component Analysis (PCA)

- **PCA** approximately expresses each data point $x_i \in \mathbb{R}^d$ as a linear combination of k d-dimensional vectors $u_1, \ldots, u_r \in \mathbb{R}^d$, such that

$$x_i \approx \bar{x} + a_i[1] \cdot u_1 + \cdots + a_i[r] \cdot u_r$$

- **PCA** is a formal definition of the "best" such vectors $u_1, \cdots, u_r$.
- **Pre-processing**:
  - To avoid keeping track of the mean vector $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$ and make the following linear algebra simple, we will assume that the data is pre-processed to remove the mean such that $\bar{x} = 0$.

Real example: eigen-face

d=32x32pixels per image

n images of faces

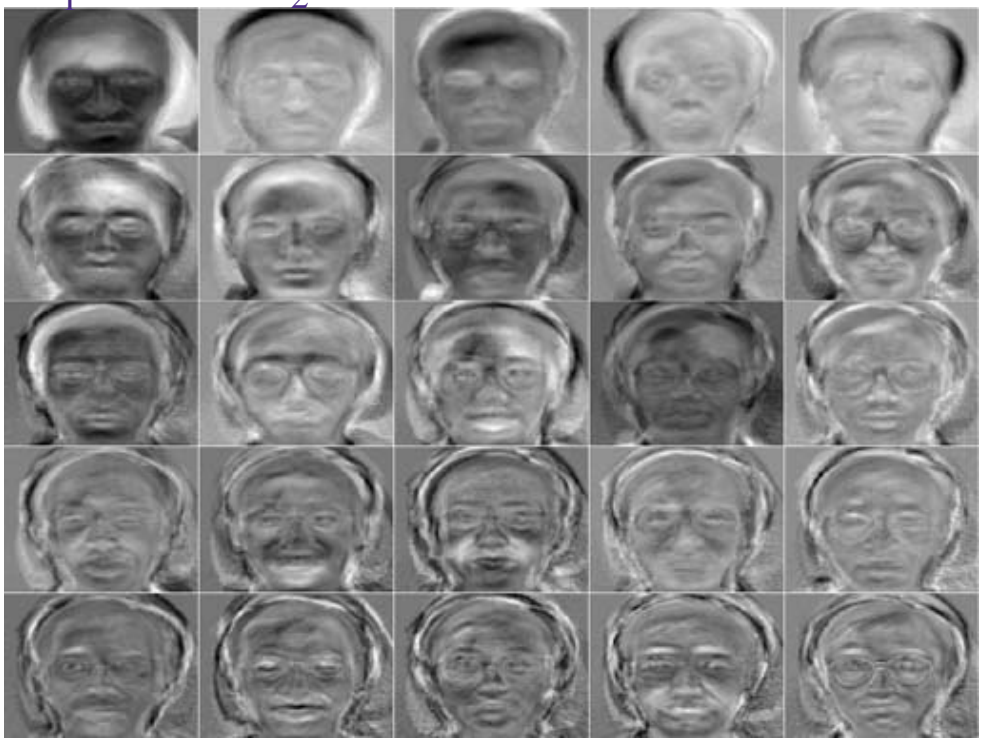
$x_1 =$

Principal component analysis finds a compact linear representation

- **patterns** that capture the distinct features of the samples is called **principal component** (to be formally defined later)
- we use $r = 25$ principal components

Principal components:

$u_1 \in \mathbb{R}^d$ u_2

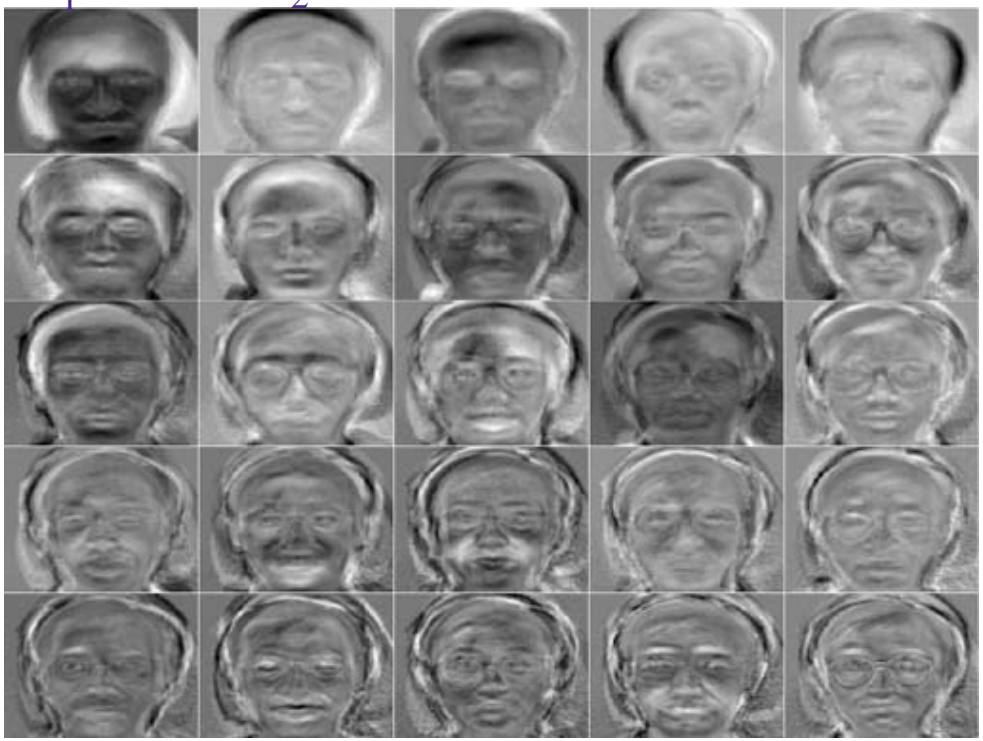


Principal component analysis finds a compact linear representation

- **patterns** that capture the distinct features of the samples is called **principal component** (to be formally defined later)
- we use $r = 25$ principal components
- we can represent each sample as a **weighted linear combination** of the principal components, and just store the weights (as opposed to all pixel values)

Principal components:

$u_1 \in \mathbb{R}^d$ u_2



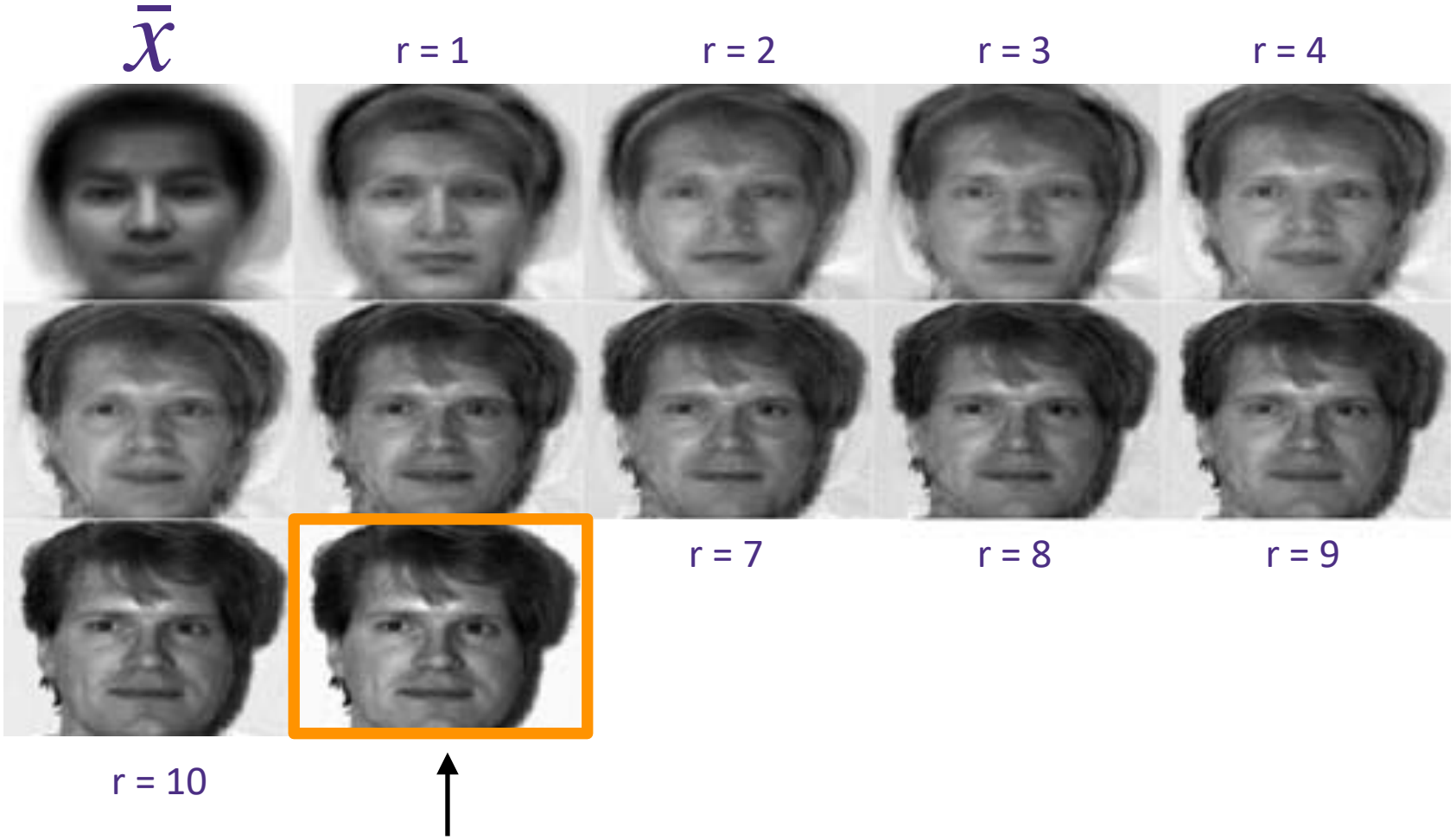
$$\approx a[1]u_1 + a[2]u_2 + \cdots + a[25]u_{25}$$

- Each image is now represented by $r = 25$ numbers $a = (a[1], \ldots, a[25])$
- To store n images, it requires memory of only $d \times r + r \times n \ll d \times n$

$1{,}000 \times 25 + 25 \times n$ $\quad$ $1{,}000 \times n$

10 principal components give a pretty good reconstruction of a face

average face $\bar{x} + a[1]u_1$ $\bar{x} + a[1]u_1 + a[2]u_2$



Ground truths real face

Assumption

- Notice how we started with the average face $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
- PCA is applied to pre-processed data $\{x_i - \bar{x}\}_{i=1}^{n}$
- For simplicity, we will assume that x_i's are centered such that $\frac{1}{n}\sum_{i=1}^{n} x_i = 0$
- otherwise, without loss of generality,
  everything we do can be applied to the re-centered version of the data,
  i.e. $\{x_i - \bar{x}\}_{i=1}^{n}$, with $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$

How do we define the principal components?

- Dimensionality reduction (for some $r \ll d$):
  we would like to have a set of orthogonal directions $u_1, \ldots, u_r \in \mathbb{R}^d$, with $\|u_j\|_2 = 1$ for all j to uniquely define principal components when we can, such that each data can be represented as linear combination of those direction vectors, i.e.

$$x_i \approx p_i = a_i[1]u_1 + \cdots + a_i[r]u_r$$

d=32x32

$$x_i = \begin{bmatrix} x_i[1] \\ \vdots \\ x_i[d] \end{bmatrix} \xrightarrow[\text{Reduction}]{\text{Dimensionality}} a_i = \begin{bmatrix} a_i[1] \\ \vdots \\ a_i[r] \end{bmatrix}$$

- Which choice of the principal components, $\{u_1, \ldots, u_r\}$, are better?
- But first, how do we find a_i for a data point x_i given $\{u_1, \ldots, u_r\}$?

How do we find the principal components?

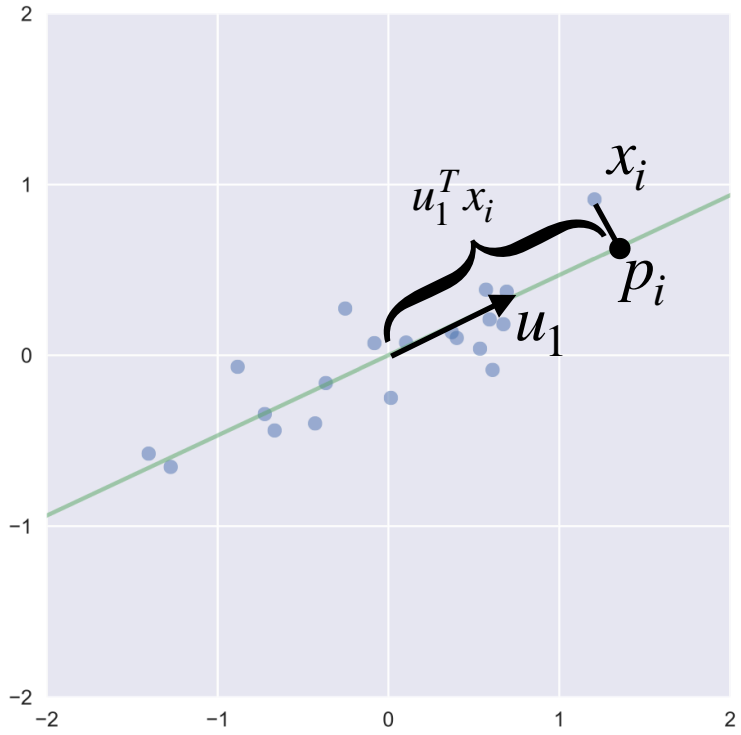
- Dimensionality reduction (for some $r \ll d$):
  we would like to have a set of orthogonal directions $u_1, \ldots, u_r \in \mathbb{R}^d$, with $\|u_j\|_2 = 1$ for all j, such that each data can be represented as linear combination of those direction vectors, i.e.

$$x_i \approx p_i = a_i[1]u_1 + \cdots + a_i[r]u_r$$

- **given a set of $u_1, \ldots, u_r$ with $\|u_j\| = 1$,**
  the best representation p_i of x_i
  is the projection of the point onto
  the subspace spanned by u_j's, i.e.

$$a_i[j] = u_j^T x_i$$

$$p_i = \sum_{j=1}^{r} \underbrace{(u_j^T x_i)}_{a_i[j]} u_j$$



How do we find the principal components?

- Dimensionality reduction (for some $r \ll d$):
  we would like to have a set of orthogonal directions $u_1, \ldots, u_r \in \mathbb{R}^d$, with $\|u_j\|_2 = 1$ for all j, such that each data can be represented as linear combination of those direction vectors, i.e.

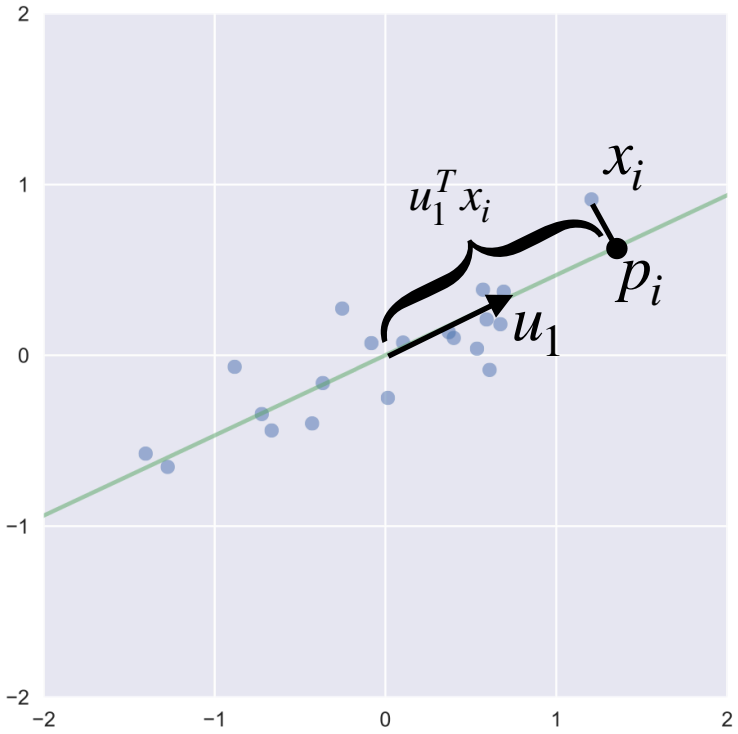
$$x_i \approx p_i = a_i[1]u_1 + \cdots + a_i[r]u_r$$

- **given a set of $u_1, \ldots, u_r$ with $\|u_j\| = 1$,**
  the best representation p_i of x_i
  is the projection of the point onto
  the subspace spanned by u_j's, i.e.

$$a_i[j] = u_j^T x_i$$

$$p_i = \sum_{j=1}^{r} \underbrace{(u_j^T x_i)}_{a_i[j]} u_j$$

- proof for a single u_1: $\min_a \|x_i - a\, u_1\|^2$
- How do we find the u_j's?



Principal components is the subspace that minimizes the reconstruction error

$$\underset{u_1,\ldots,u_r}{\text{minimize}} \quad \frac{1}{n}\sum_{i=1}^{n} \|x_i - p_i\|_2^2$$

subject to $\|u_j\|_2 = 1$ for all j and $u_j^T u_\ell = 0$ for all $j \neq \ell$

$$p_i = \sum_{j=1}^{r} (u_j^T x_i) u_j = \sum_{j=1}^{r} u_j u_j^T x_i = \left(\sum_{j=1}^{r} u_j u_j^T\right) x_i = \mathbf{U}\mathbf{U}^T x_i$$

where $\mathbf{U} = [u_1 \quad u_2 \quad \cdots \quad u_r] \in \mathbb{R}^{d \times r}$

Principal components is the subspace that minimizes the reconstruction error

$$\underset{u_1,\ldots,u_r}{\text{minimize}} \ \frac{1}{n}\sum_{i=1}^{n} \|x_i - p_i\|_2^2$$

$$\text{subject to} \ \ \|u_j\|_2 = 1 \text{ for all } j \text{ and } u_j^T u_\ell = 0 \text{ for all } j \neq \ell$$

$$p_i = \sum_{j=1}^{r} (u_j^T x_i) u_j = \sum_{j=1}^{r} u_j u_j^T x_i = \left(\sum_{j=1}^{r} u_j u_j^T\right) x_i = \mathbf{U}\mathbf{U}^T x_i$$

where $\mathbf{U} = [u_1 \quad u_2 \quad \cdots \quad u_r] \in \mathbb{R}^{d \times r}$

$$\underset{U}{\text{minimize}} \ \frac{1}{n}\sum_{i=1}^{n} \|x_i - \mathbf{U}\mathbf{U}^T x_i\|_2^2$$

$$\text{subject to} \ \ \mathbf{U}^T\mathbf{U} = \mathbf{I}_{r\times r}$$

- Small rank r gives efficiency and large r gives less reconstruction error
- Q. How do we solve this optimization?

Minimizing reconstruction error to find principal components

$$\underset{U}{\text{minimize}} \ \frac{1}{n}\sum_{i=1}^{n} \|x_i - \mathbf{U}\mathbf{U}^T x_i\|_2^2$$

subject to $\mathbf{U}^T\mathbf{U} = \mathbf{I}_{r\times r}$

Minimizing reconstruction error to find principal components

$$\frac{1}{n}\sum_{i=1}^{n}\|x_i - UU^T x_i\|_2^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left\{\|x_i\|_2^2 - 2x_i^T UU^T x_i + x_i^T U \underbrace{U^T U}_{=\mathbf{I}} U^T x_i\right\}$$

$$= \underbrace{\frac{1}{n}\sum_{i=1}^{n}\|x_i\|_2^2}_{\text{does not depend on } U} - \frac{1}{n}\sum_{i=1}^{n} x_i^T UU^T x_i$$

$$= C - \sum_{j=1}^{r} \underbrace{\frac{1}{n}\sum_{i=1}^{n}(u_j^T x_i)^2}_{\text{Variance in direction } u_j}$$

Recall we assumed x_i's are centered, i.e., zero-mean

Minimize Reconstruction Error

$$\underset{U}{\text{minimize}} \ \frac{1}{n}\sum_{i=1}^{n}\|x_i - \mathbf{U}\mathbf{U}^T x_i\|_2^2$$

$$\text{subject to } \mathbf{U}^T\mathbf{U} = \mathbf{I}_{r\times r}$$

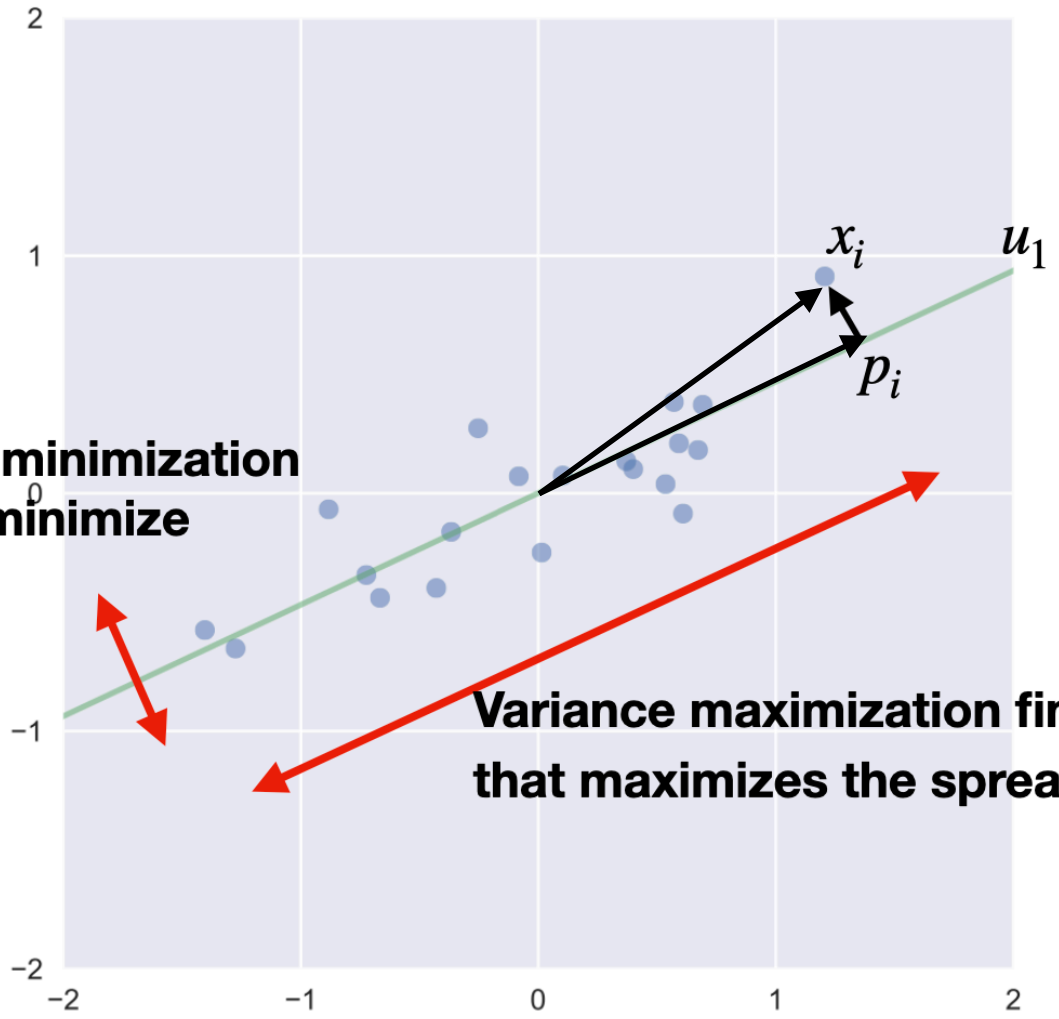
Maximizing Variance captured in principal directions

$$\underset{U}{\text{maximize}} \ \sum_{j=1}^{r}\frac{1}{n}\sum_{i=1}^{n}(u_j^T x_i)^2$$

$$\text{subject to } \mathbf{U}^T\mathbf{U} = \mathbf{I}_{r\times r}$$

Variance maximization vs. reconstruction error minimization

- both give the same principal components as optimal solution, because by Pythagorean Theorem: Error^2 + Variance of data $= \|x_i\|_2^2$



Reconstruction error minimization finds directions that minimize the distances to p_i's

Variance maximization finds directions that maximizes the spread of p_i's

Maximizing variance to find principal components

$$\underset{U}{\text{maximize}} \quad \sum_{j=1}^{r} \frac{1}{n} \sum_{i=1}^{n} (u_j^T x_i)^2$$

$$\text{subject to} \quad \mathbf{U}^T\mathbf{U} = \mathbf{I}_{r\times r}$$

We will solve it for $r = 1$ case,
and the general case follows similarly

$$\underset{u:\|u\|_2=1}{\text{maximize}} \quad \frac{1}{n} \sum_{i=1}^{n} (u^T x_i)^2$$

$$\underset{u:\|u\|_2=1}{\text{maximize}} \quad u^T C u$$

How do you find u?

Maximizing variance to find principal components

$$\text{maximize}_u \; u^T \mathbf{C} u \qquad (a)$$

$$\textbf{subject to} \quad \|u\|_2^2 = 1$$

- we first claim that this optimization problem has the same optimal solution as the following **inequality constrained** problem

$$\text{maximize}_u \; u^T \mathbf{C} u \qquad (b)$$

$$\textbf{subject to} \quad \|u\|_2^2 \leq 1$$

- Why?

Maximizing variance to find principal components

$$\text{maximize}_u \; u^T \mathbf{C} u \qquad (a)$$

$$\textbf{subject to} \quad \|u\|_2^2 = 1$$

- we first claim that this optimization problem has the same optimal solution as the following **inequality constrained** problem

$$\text{maximize}_u \; u^T \mathbf{C} u \qquad (b)$$

$$\textbf{subject to} \quad \|u\|_2^2 \leq 1$$

- the reason is that, because $u^T \mathbf{C} u \geq 0$ for all $u \in \mathbb{R}^d$, the optimal solution of (b) has to have $\|u\|_2^2 = 1$
- if it did not have $\|u\|_2^2 = 1$, say $\|u\|_2^2 = 0.9$, then we can just multiply this u by a constant factor of $\sqrt{10/9}$ and increase the objective by a factor of $10/9$ while still satisfying the constraints
- Why is (b) better than (a)?

$$\text{maximize}_u \; u^T \mathbf{C} u \qquad (b)$$

$$\textbf{subject to} \quad \|u\|_2^2 \leq 1$$

- we are maximizing the variance, while **keeping u small**
- this can be reformulated as an unconstrained problem, with Lagrangian encoding, to move the constraint into the objective

$$\text{maximize}_{u \in \mathbb{R}^d} \quad \underbrace{u^T \mathbf{C} u - \lambda \|u\|_2^2}_{F_\lambda(u)} \qquad (c)$$

- this encourages small u as we want, and we can make this connection precise: there exists a (unknown) choice of λ such that the optimal solution of (c) is the same as the optimal solution of (b)
- further, for this choice of λ, exists an optimal u^* with $\|u^*\|_2 = 1$

Solving the unconstrained optimization

$$\text{maximize}_{u \in \mathbb{R}^d} \quad \underbrace{u^T \mathbf{C} u - \lambda \|u\|_2^2}_{F_\lambda(u)}$$

- to find such λ and the corresponding u, we solve the unconstrained optimization, by setting the gradient to zero

$$\nabla F_\lambda(u) \;=\; 2\mathbf{C}u - 2\lambda u \;\;=\; 0$$

- the candidate solution satisfies: $\mathbf{C}u = \lambda u$,
  i.e. an **eigenvector** of $\mathbf{C}$
- All eigenvalue-eigenvector pairs $\{(\lambda^{(i)}, u^{(i)})\}_{i=1}^{d}$ of $\mathbf{C}$ satisfy the above equation (and we will only consider normalized $\|u^{(i)}\| = 1$)
- We know that $\lambda^{(i)} \geq 0$, because $\mathbf{C}$ is positive semidefinite
- We assume these are ordered such that $\lambda^{(1)} \geq \lambda^{(2)} \geq \cdots \geq \lambda^{(d)}$
- We only need to consider the eigenvectors in maximizing the original objective: $u^T \mathbf{C} u$, which is maximized by the first eigenvector $u^{(1)}$

The principal component analysis

- so far we considered finding ONE principal component $u \in \mathbb{R}^d$
- it is the eigenvector corresponding to the maximum eigenvalue of the covariance matrix

$$\mathbf{C} = \frac{1}{n}\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{d \times d}$$

- We can also use the Singular Value Decomposition (SVD) to find such an eigenvector
- note that is the data is not centered at the origin, we should re-center the data before applying SVD

- in general we define and use multiple principal components
- if we need r principal components, we take r eigenvectors corresponding to the **largest r eigenvalues** of $\mathbf{C}$

Algorithm: Principal Component Analysis

- **input**: data points $\{x_i\}_{i=1}^n$, target dimension $r \ll d$
- **output**: r-dimensional subspace U
- **algorithm**:
  - compute mean $\quad \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$
  - compute covariance matrix

$$\mathbf{C} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(x_i - \bar{x})^T$$

  - let $(u_1, \ldots, u_r)$ be the set of (normalized) eigenvectors with corresponding to the largest r eigenvalues of $\mathbf{C}$
  - return $\mathbf{U} = [u_1 \quad u_2 \quad \cdots \quad u_r]$

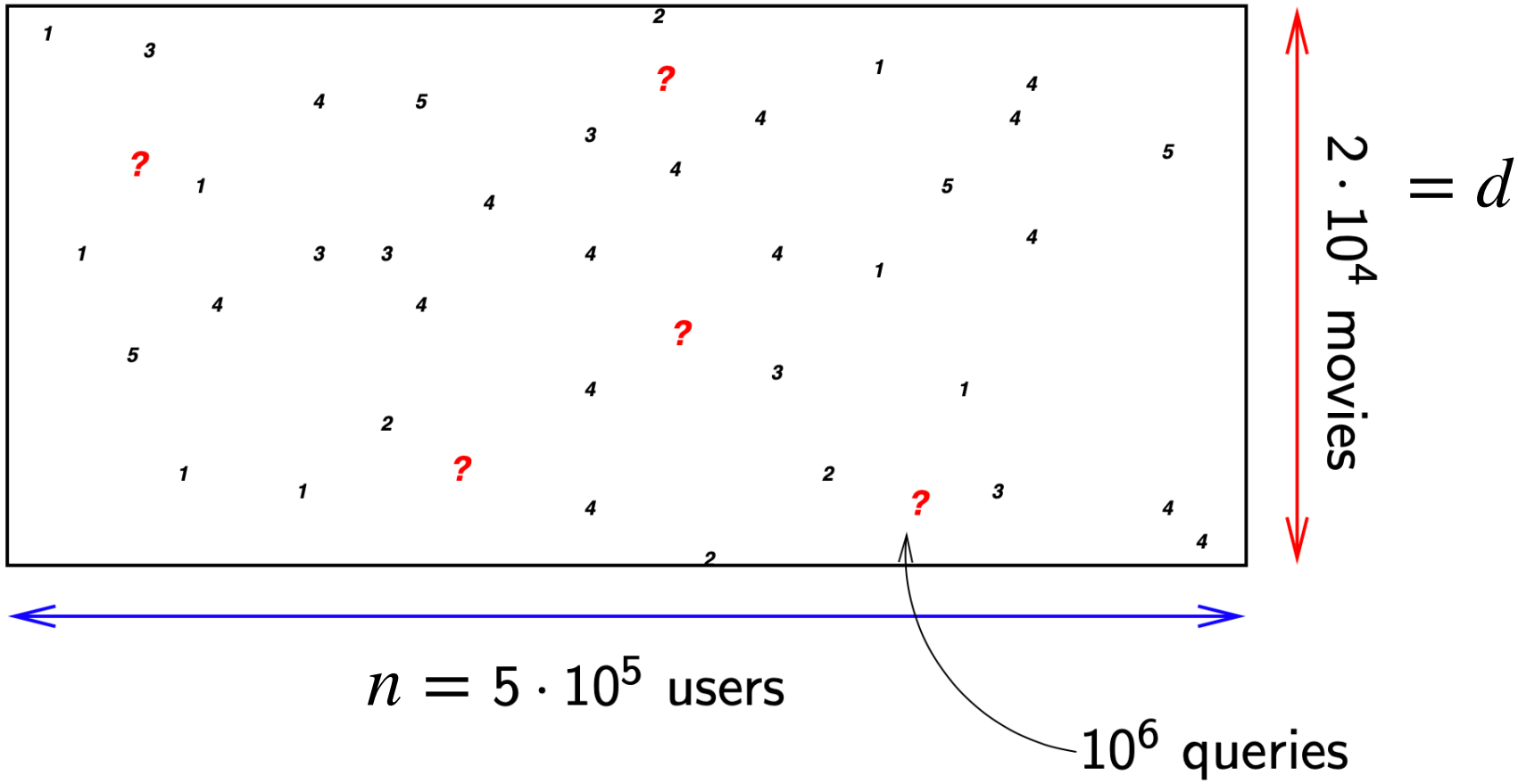
- further the data points can be represented compactly via

$$a_i = \mathbf{U}^T(x_i - \bar{x}) \in \mathbb{R}^r \text{ and}$$

each data point approximated by

$$p_i = \bar{x} + U\, a_i$$

Matrix completion for recommendation systems



- users provide ratings on a few movies, and we want to predict the missing entries in this ratings matrix, so that we can make recommendations
- without any assumptions, the missing entries can be anything, and no prediction is possible

Matrix completion

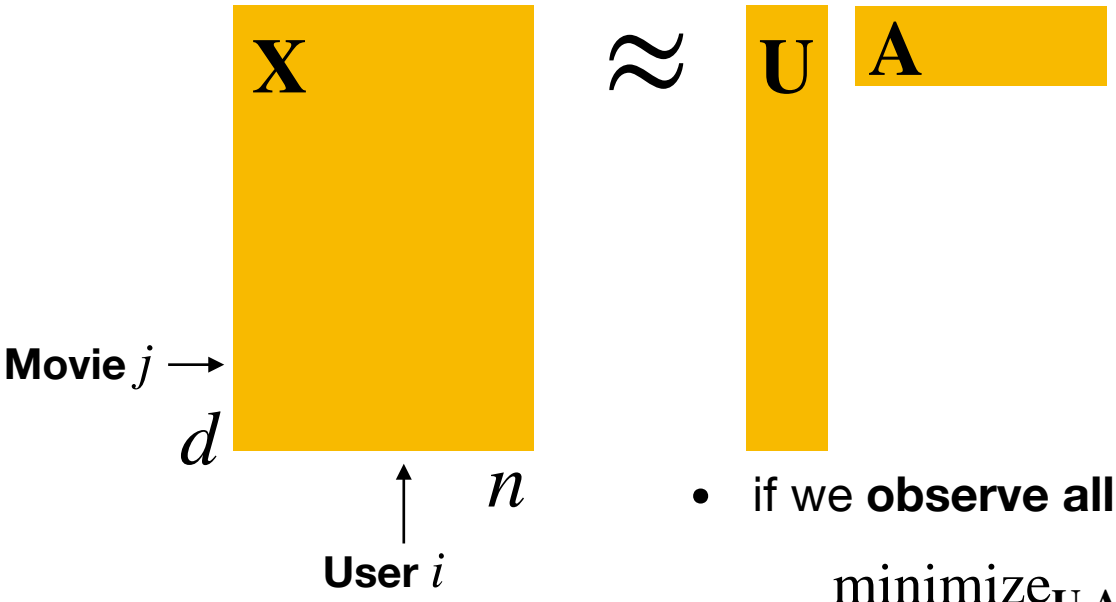
- however, the ratings are not arbitrary, but people with similar tastes rate similarly
- such structure can be modeled using low dimensional representation of the data as follows
- we will find a set of principal component vectors
  $\mathbf{U} = [u_1 \quad u_2 \quad \cdots \quad u_r] \in \mathbb{R}^{d \times r}$
- such that that ratings $x_i \in \mathbb{R}^d$ of user i, can be represented as

$$\begin{aligned} x_i &= a_i[1]u_1 + \cdots a_i[r]u_r \\ &= \mathbf{U}a_i \end{aligned}$$

  for some lower-dimensional $a_i \in \mathbb{R}^r$ for i-th user and some $r \ll d$
- for example, $u_1 \in \mathbb{R}^d$ means how horror movie fans like each of the d movies,
- and $a_i[1]$ means how much user i is fan of horror movies

Matrix completion

- let $\mathbf{X} = [x_1 \quad x_2 \quad \cdots \quad x_n] \in \mathbb{R}^{d \times n}$ be the ratings matrix, and assume it is fully observed, i.e. we know all the entries
- then we want to find $\mathbf{U} \in \mathbb{R}^{d \times r}$ and $\mathbf{A} = [a_1 \quad a_2 \quad \cdots \quad a_n] \in \mathbb{R}^{r \times n}$ that approximates $\mathbf{X}$



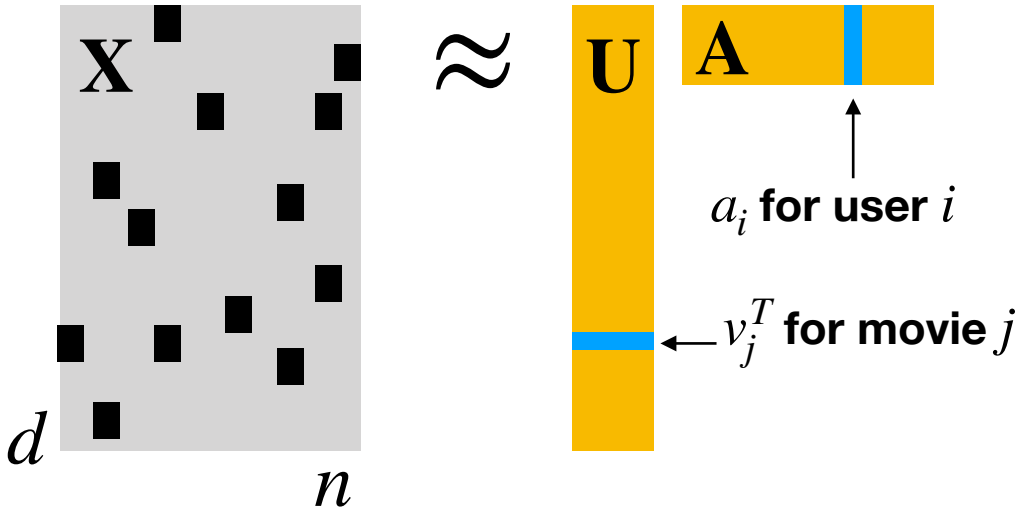
- if we **observe all entries** of $\mathbf{X}$, then we can solve

$$\text{minimize}_{\mathbf{U},\mathbf{A}} \sum_{i=1}^{n} \|x_i - \mathbf{U}a_i\|_2^2$$

which can be solved using PCA (i.e. SVD)

Matrix completion

- in practice, we only observe $\mathbf{X}$ partially
- let $S_{\text{train}} = \{(i_\ell, j_\ell)\}_{\ell=1}^{N}$ denote N observed ratings for user i_ℓ on movie j_ℓ



- let v_j^T denote the j-th row of $\mathbf{U}$ and a_i denote i-th column of $\mathbf{A}$
- then user i's rating on movie j, i.e. $\mathbf{X}_{ji}$ is approximated by $v_j^T a_i$, which is the inner product of v_j (a column vector) and a column vector a_i
- we can also write it as $\langle v_j, a_i \rangle = v_j^T a_i$

Matrix completion

- a natural approach to fit v_j's and $a_i's$ to given training data is to solve

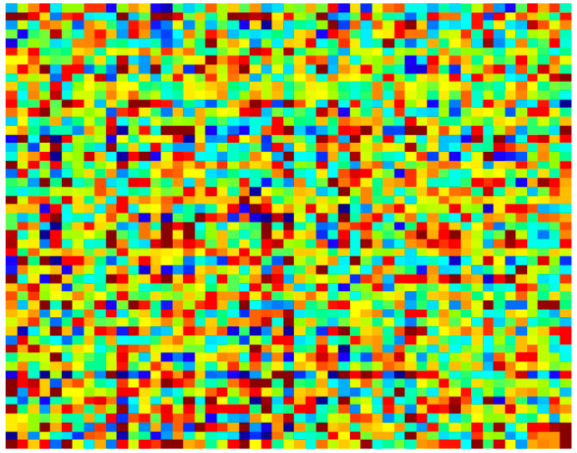
$$\text{minimize}_{\mathbf{U},\mathbf{A}} \sum_{(i,j)\in S_{\text{train}}} (\mathbf{X}_{ji} - v_j^T a_i)^2$$

- this can be solved, for example via gradient descent or alternating minimization
- this can be quite accurate, with small number of samples

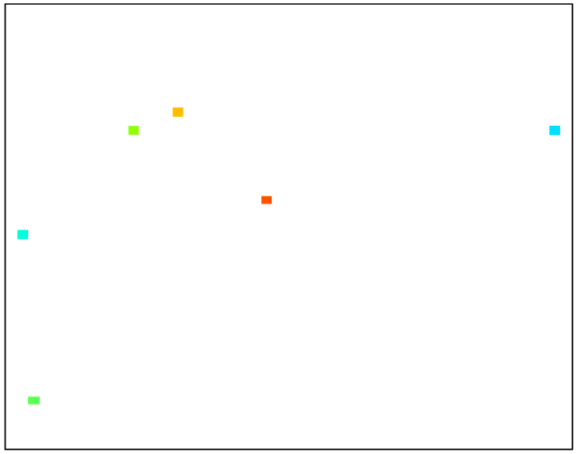
Example: 2000×2000 rank-8 random matrix

For illustration, we zoom in to a 50x50 submatrix

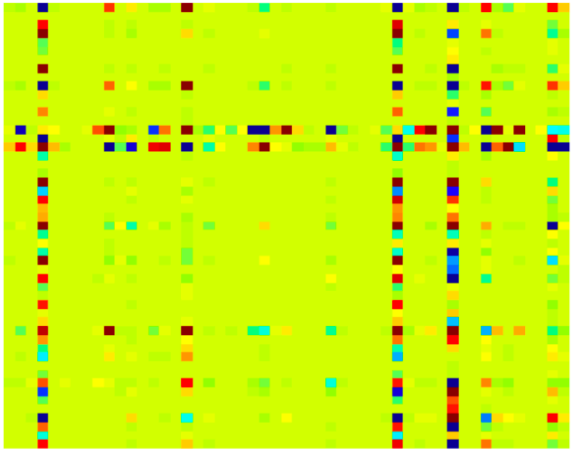
low-rank matrix $\mathbf{X}$



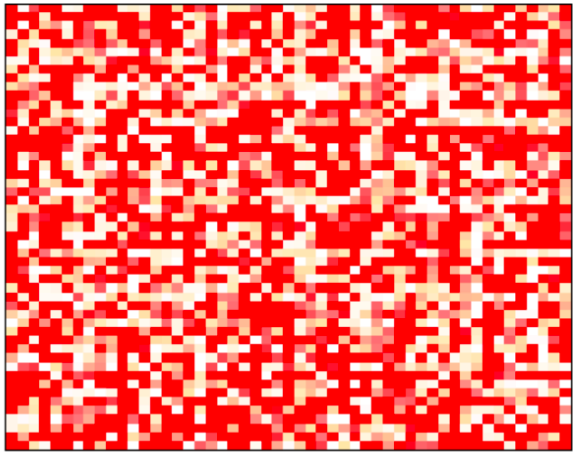
sampled matrix



Gradient descent output $\mathbf{UA}$



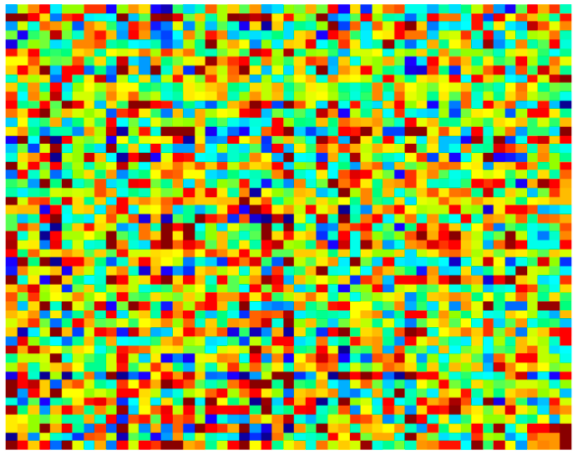
squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



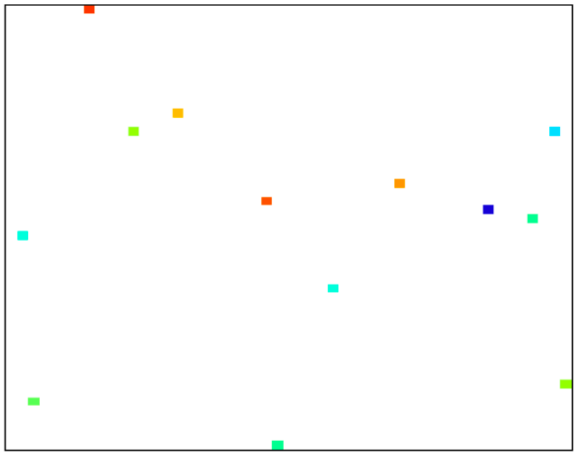
0.25% sampled

Example: 2000×2000 rank-8 random matrix

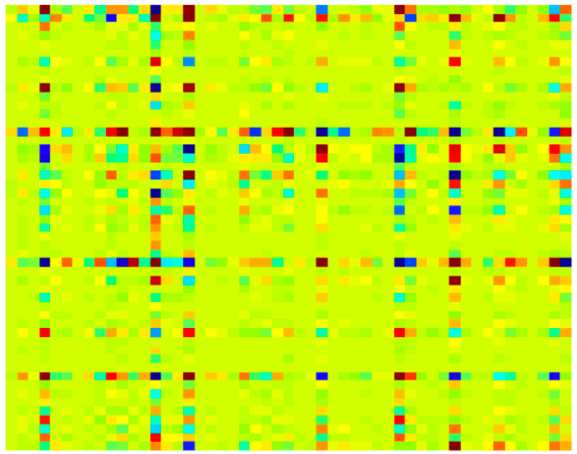
low-rank matrix $\mathbf{X}$



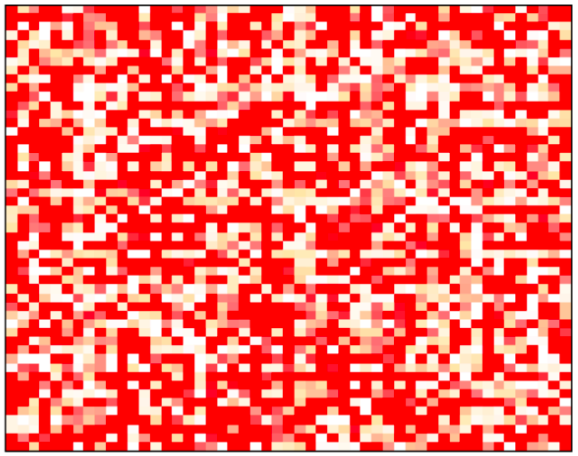
sampled matrix



Gradient descent output $\mathbf{UA}$



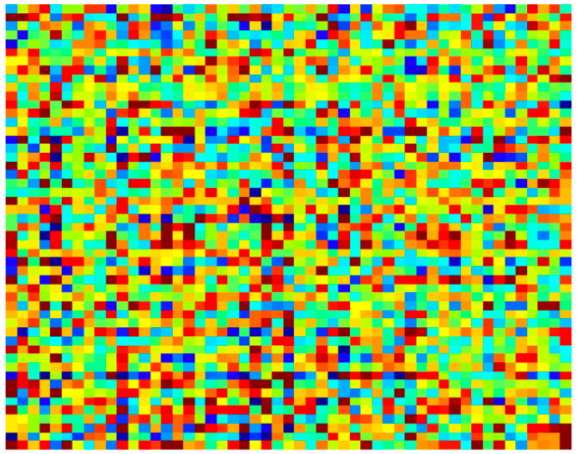
squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



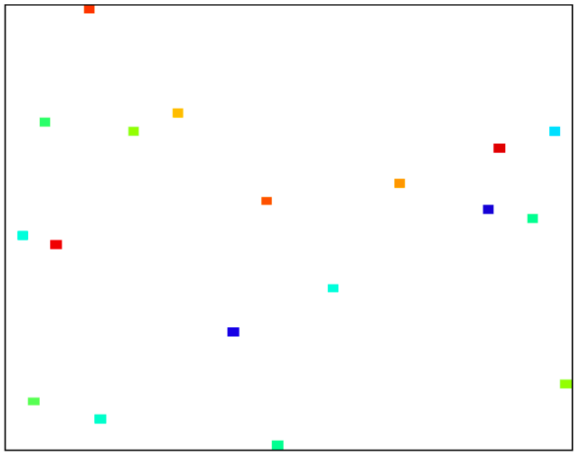
0.50% sampled

Example: 2000×2000 rank-8 random matrix

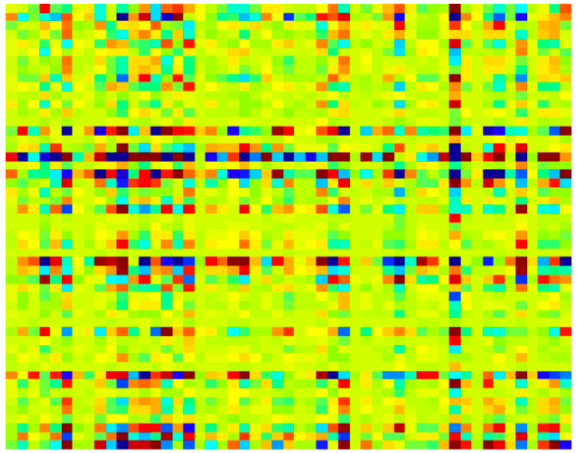
low-rank matrix $\mathbf{X}$



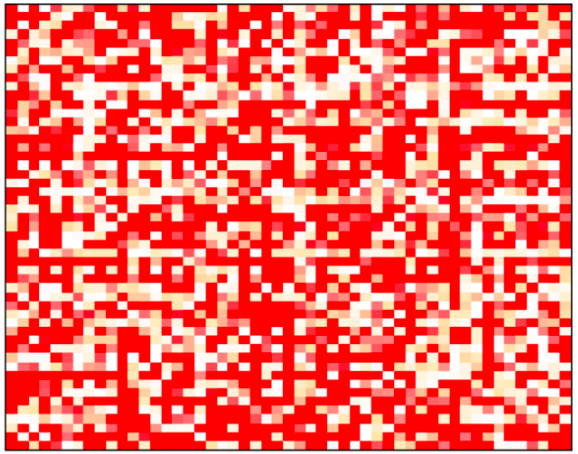
sampled matrix



Gradient descent output $\mathbf{UA}$



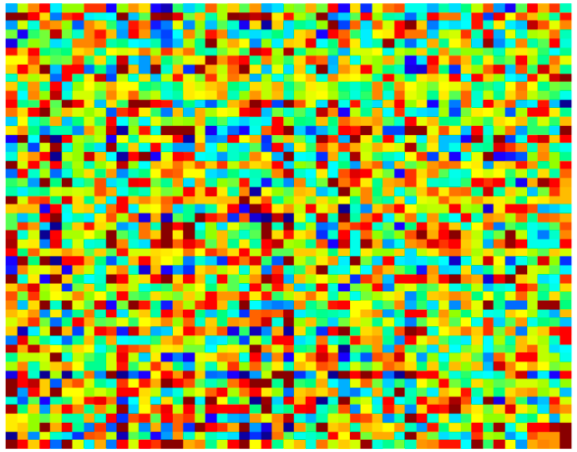
squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



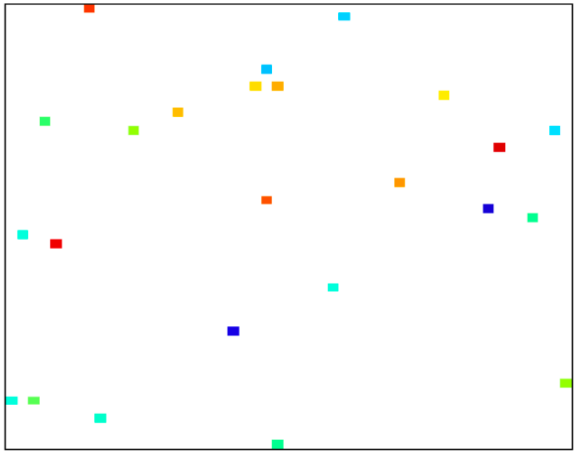
0.75% sampled

Example: 2000×2000 rank-8 random matrix

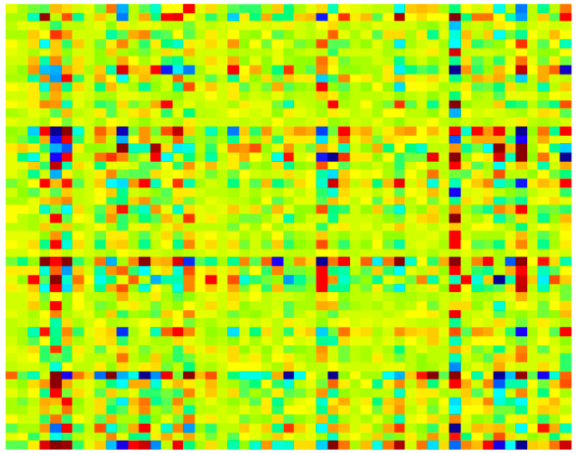
low-rank matrix $\mathbf{X}$



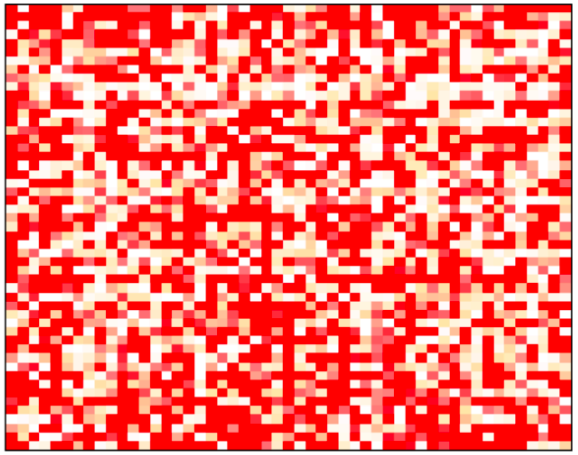
sampled matrix



Gradient descent output $\mathbf{UA}$



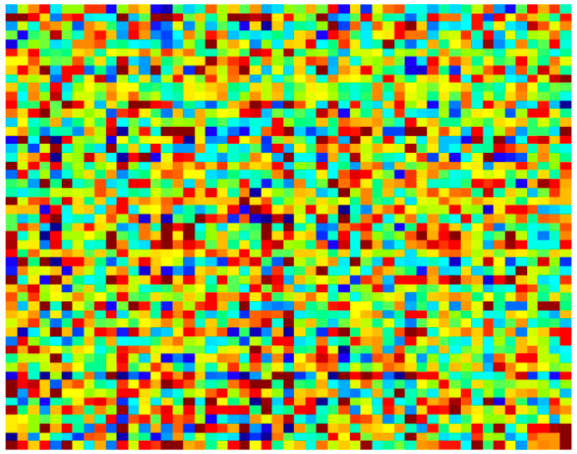
squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



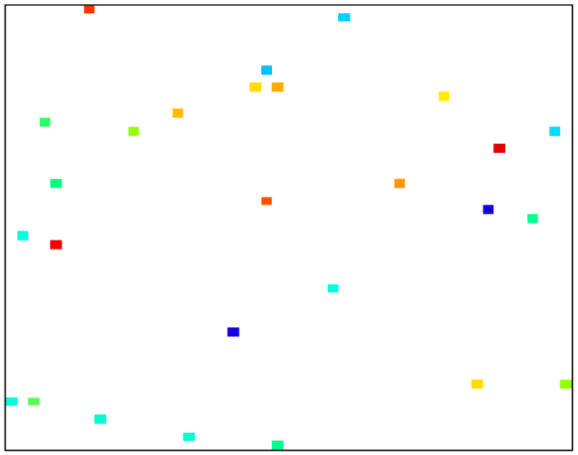
1.00% sampled

Example: 2000×2000 rank-8 random matrix

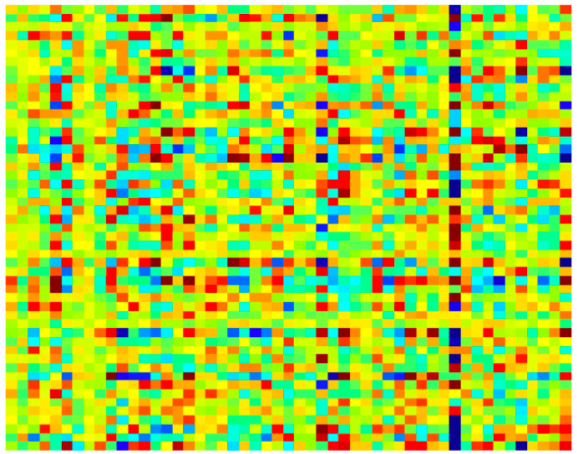
low-rank matrix $\mathbf{X}$



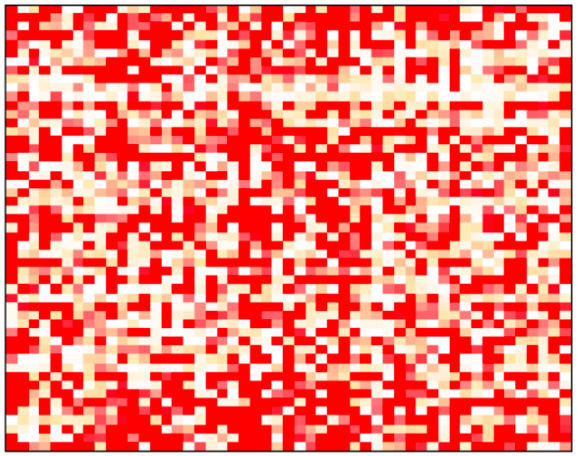
sampled matrix



Gradient descent output $\mathbf{UA}$



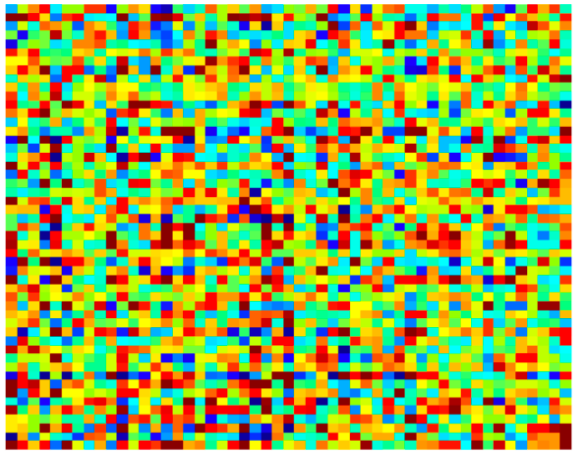
squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



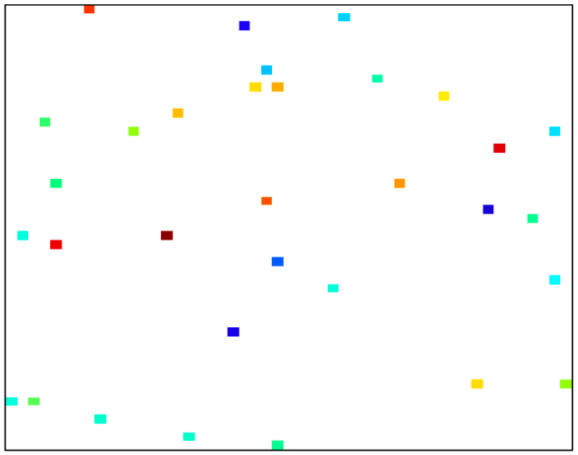
1.25% sampled

Example: 2000×2000 rank-8 random matrix

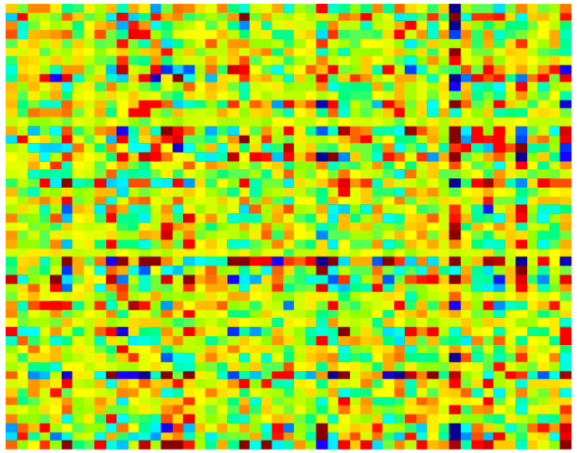
low-rank matrix $\mathbf{X}$



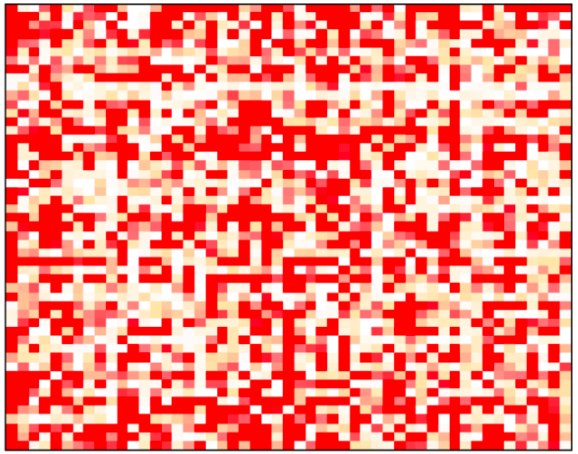
sampled matrix



Gradient descent output $\mathbf{UA}$



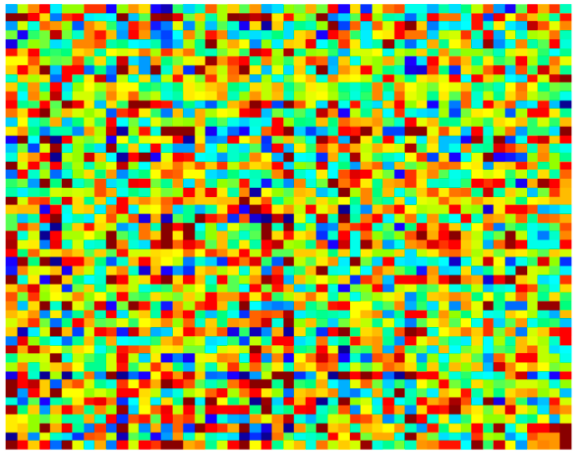
squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$



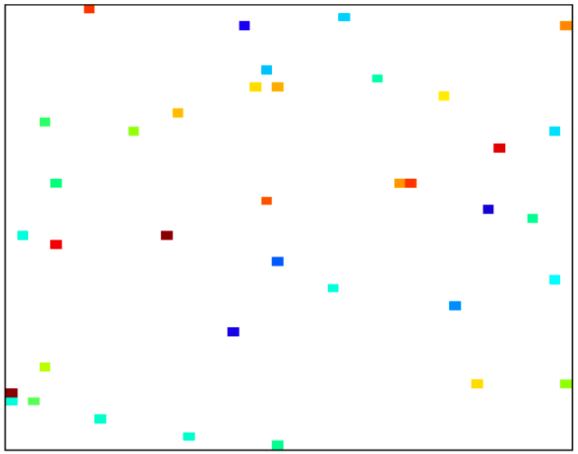
1.50% sampled

Example: 2000×2000 rank-8 random matrix

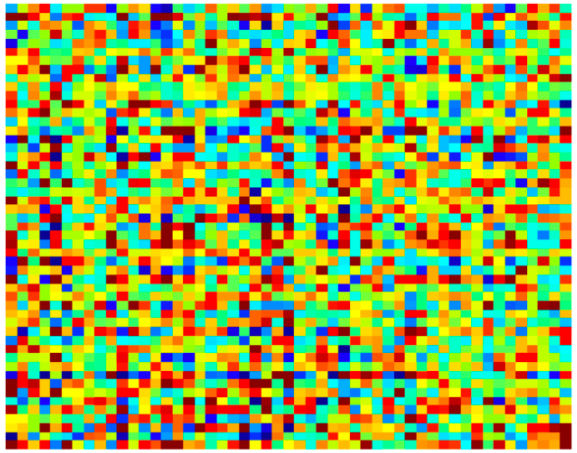
low-rank matrix $\mathbf{X}$



sampled matrix



Gradient descent output $\mathbf{UA}$



squared error $(\mathbf{X}_{ji} - (\mathbf{UA})_{ji})^2$

1.75% sampled

Questions?