Matrix Factorisation with Linear Filters

Ömer Deniz Akyıldız

GTS Research Seminar

October 22, 2015

outline

introduction, notation, and other things

a brief review of linear filters

matrix factorisation as a linear filtering problem

application to image restoration

some afterthoughts

introduction, notation, and other things

Formally, matrix factorization is the problem of factorizing a data matrix $Y \in \mathbb{R}^{m \times n}$ into,

$$Y \approx CX \tag{1}$$

where $C \in \mathbb{R}^{m \times r}$ and $X \in \mathbb{R}^{r \times n}$. Here r is the approximation rank which is typically selected by hand. These methods can be interpreted as dictionary learning:

- columns of C define the elements of the dictionary,
- columns of X can be thought as associated coefficients.

$$\underbrace{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}}_{Y} \approx \underbrace{\begin{bmatrix} \times & \times \\ \times & \times \\ \times & \times \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}}_{X}$$

The classical computation of MFs:

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \approx$$

The classical computation of MFs:

Update C by fixing X and using the whole dataset.

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \approx \begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}$$

The classical computation of MFs:

Update X by fixing C and using the whole dataset.

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \approx \begin{bmatrix} \times & \times \\ \times & \times \\ \times & \times \end{bmatrix} \begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}$$

The classical computation of MFs:

Update C by fixing X and using the whole dataset.

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \approx \begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}$$

The classical computation of MFs:

Update X by fixing C and using the whole dataset.

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \approx \begin{bmatrix} \times & \times \\ \times & \times \\ \times & \times \end{bmatrix} \begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}$$

The classical computation of MFs:

But what if?

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \end{bmatrix}$$

The classical computation of MFs:

But what if?

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \end{bmatrix}$$

The classical computation of MFs:

But what if?

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times & \times \end{bmatrix}$$

The classical computation of MFs:

But what if?

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times & \times & \times & \cdots & \infty \\ \times & \times & \times & \times & \times & \times & \times & \times & \cdots & \infty \\ \times & \times & \times & \times & \times & \times & \times & \times & \cdots & \infty \end{bmatrix}$$

introduction, notation, and other things

goal of this work

$$Y \approx CX$$

$Y \in \mathbb{R}^{m\times n}$, $C \in \mathbb{R}^{m\times r}$, and $X \in \mathbb{R}^{r\times n}$.

- In this work, we aim to solve this problem online:
  - each column of the data matrix is an observation for us.

$$\underbrace{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}}_{Y} \approx \underbrace{\begin{bmatrix} \times & \times \\ \times & \times \\ \times & \times \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}}_{X}$$

We sample columns uniformly random.

$$\underbrace{\begin{bmatrix} \times & \boldsymbol{\times} & \times & \times & \times \\ \times & \boldsymbol{\times} & \times & \times & \times \\ \times & \underbrace{\boldsymbol{\times}}_{y_1} & \times & \times & \times \end{bmatrix}}_{Y} \approx \underbrace{\begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}}_{C_1} \underbrace{\begin{bmatrix} \times & \boldsymbol{\times} & \times & \times & \times \\ \times & \underbrace{\boldsymbol{\times}}_{x_1} & \times & \times & \times \end{bmatrix}}_{X}$$

We sample columns uniformly random.

$$\underbrace{\begin{bmatrix} \times & \times & \boldsymbol{\times} & \times & \times \\ \times & \times & \boldsymbol{\times} & \times & \times \\ \times & \times & \underbrace{\boldsymbol{\times}}_{y_2} & \times & \times \end{bmatrix}}_{Y} \approx \underbrace{\begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}}_{C_2} \underbrace{\begin{bmatrix} \times & \times & \boldsymbol{\times} & \times & \times \\ \times & \times & \underbrace{\boldsymbol{\times}}_{x_2} & \times & \times \end{bmatrix}}_{X}$$

We sample columns uniformly random.

$$\underbrace{\begin{bmatrix} \underbrace{\begin{matrix} \boldsymbol{\times} \\ \boldsymbol{\times} \\ \boldsymbol{\times} \end{matrix}}_{y_3} & \begin{matrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{matrix} \end{bmatrix}}_{Y} \approx \underbrace{\begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}}_{C_3} \underbrace{\begin{bmatrix} \underbrace{\begin{matrix} \boldsymbol{\times} \\ \boldsymbol{\times} \end{matrix}}_{x_3} & \begin{matrix} \times & \times & \times & \times \\ \times & \times & \times & \times \end{matrix} \end{bmatrix}}_{X}$$

We sample columns uniformly random.

$$\underbrace{\begin{bmatrix} \times & \times & \times & \times & \boldsymbol{\times} \\ \times & \times & \times & \times & \boldsymbol{\times} \\ \times & \times & \times & \times & \underbrace{\boldsymbol{\times}}_{y_4} \end{bmatrix}}_{Y} \approx \underbrace{\begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}}_{C_4} \underbrace{\begin{bmatrix} \times & \times & \times & \times & \boldsymbol{\times} \\ \times & \times & \times & \times & \underbrace{\boldsymbol{\times}}_{x_4} \end{bmatrix}}_{X}$$

introduction, notation, and other things

A note on dimensions:

- m is usually around a few thousands (but can be much more), $m = 4096$ in the experiment.
- n is basically unlimited since we process the data online.
- r is the approximation rank. In the experiment, it is 40.

For the future reference note that $mr = 163840$.

introduction, notation, and other things

In this talk,

- I_m is $m \times m$ identity matrix.
- Small letters are vectors or scalars, big letters are matrices.
  - There is no notational distinction between random or deterministic vectors/matrices.
- $\text{vec}(\cdot)$ is the vectorisation operator.
  - Example: $c = \text{vec}(C)$. If $C \in \mathbb{R}^{m \times r}$ then $c \in \mathbb{R}^{mr}$.
  - Reversion: $C = \text{vec}^{-1}_{m \times r}(c)$
- $\otimes$ denotes the Kronecker product of matrices.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} = \begin{bmatrix} 1 \cdot 0 & 1 \cdot 5 & 2 \cdot 0 & 2 \cdot 5 \\ 1 \cdot 6 & 1 \cdot 7 & 2 \cdot 6 & 2 \cdot 7 \\ 3 \cdot 0 & 3 \cdot 5 & 4 \cdot 0 & 4 \cdot 5 \\ 3 \cdot 6 & 3 \cdot 7 & 4 \cdot 6 & 4 \cdot 7 \end{bmatrix}$$

- $\odot$ denotes the Hadamard product (element-wise).

$$\mathrm{vec}(BAX) = (X^\top \otimes B)\mathrm{vec}(A). \tag{11}$$

Let us note a particular case of which this identity will be useful for us,

$$Ax = \mathrm{vec}(Ax) = (x^\top \otimes I_m)\mathrm{vec}(A). \tag{12}$$

for $\dim(A) = m \times r$, $\dim(x) = r$.

Converts the linear model over x into a linear model over A.

Kronecker products also have the following mixed product property,

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD), \tag{13}$$

and the following "inversion" property,

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}. \tag{14}$$

introduction, notation, and other things

We will use a matrix normal distribution for the matrix $C \in \mathbb{R}^{m \times r}$. Let $c = \mathrm{vec}(C)$ so $c \in \mathbb{R}^{mr}$. It is given by,

$$p(c) = \mathcal{N}(c; \mu, V \otimes U)$$

where $\mu \in \mathbb{R}^{mr}$ and $\dim(V) = r \times r$ and $\dim(U) = m \times m$.

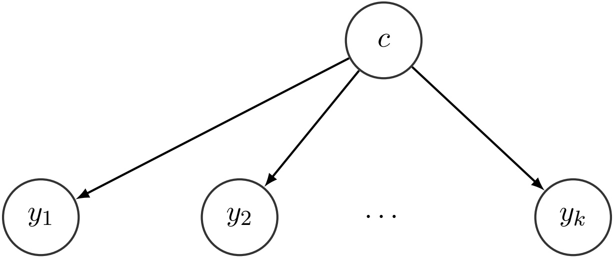
- V: Covariance matrix between columns.
- U: Covariance matrix between rows.

Since c is a $mr \times 1$ vector, covariance matrix should be $mr \times mr$ which means it can be huge (remember in our application $mr = 163840$!). But above factorisation will allow us to avoid the dimensionality problem.

a brief review of linear filters

Let us consider a generic linear probabilistic model,

$$p(c) = \mathcal{N}(c; c_0, P_0)$$
$$p(y_k|c) = \mathcal{N}(y_k; H_k c, R_k)$$



Having obtained $y_{1:k}$, we want to say something about the posterior $p(c|y_{1:k})$ via Bayesian inference.

- Recursive linear filtering: It is a simplified form of Kalman filtering – almost same recursions without a covariance prediction step.

a brief review of linear filters

It can be shown that posterior $p(c|y_{1:k})$ is Gaussian[1]:

$$p(c|y_{1:k}) = \mathcal{N}(c; c_k, P_k).$$

c_k and P_k is given by standard filtering recursions,

$$\begin{aligned}
c_k &= c_{k-1} + P_{k-1} H_k^\top (H_k P_{k-1} H_k^\top + R_k)^{-1}(y_k - H_k c_{k-1}), \\
P_k &= P_{k-1} - P_{k-1} H_k^\top (H_k P_{k-1} H_k^\top + R)^{-1} H_k P_{k-1}.
\end{aligned}$$

[1] Bayesian Filtering and Smoothing, Simo Sarkka, 2013.

matrix factorisation with linear filters

first things first: build a model

Remember the problem: $Y \approx CX$ which implies $y_k \approx Cx_k$ for any k. Let's start with this.

- We describe the following model:

$$p(y_k|c, x_k) = \mathcal{N}(y_k; Cx_k, R_k)$$

  and for simplicity choose $R_k = \lambda \otimes I_m$.

We have to put a prior on C. Matrix normal prior!

- $p(c) = \mathcal{N}(c; c_0, P_0)$ with $P_0 = V_0 \otimes I_m$ where V_0 is $r \times r$ matrix.

Why so? Tractability. Filter with full covariance $V \otimes U$ is intractable in terms of V and U.

matrix factorisation with linear filters

complete model and the big question

Now we have a probabilistic model,

$$p(c) = \mathcal{N}(c; c_0, V_0 \otimes I_m)$$
$$p(y_k|c, x_k) = \mathcal{N}(y_k; Cx_k, \lambda \otimes I_m)$$

and it comes to the big question: How can we do inference on c variable, and estimate x_k at the same time, and still hope for a reasonable computational budget?

Notice that this question is not easy to answer. $c \in \mathbb{R}^{mr}$ and mr can be large (e.g. a hundred thousand). What we do in this work is mainly that giving an approximate and fast answer.

Let us start with estimation.

matrix factorisation with linear filters

estimation

Let us deal first with the easy problem. For each "observation" y_k, we would like to estimate x_k.

The word "estimation" comes from the fact that x_k is assumed to be a deterministic but unknown quantity.

Problem formulation:

$$x_k^* = \underset{x_k}{\operatorname{argmax}}\, p(y_k | c, x_k)$$

The problem with this: We don't know c. Two options:

- Fill c with c_{k-1} ("guesstimation").
- Be a true Bayesian, and integrate out c.

It is very hard to be principled! We will go with the first option.

matrix factorisation with linear filters

estimation

“Guesstimation”

$$\begin{aligned} x_k^* &= \operatorname*{argmax}_{x_k} p(y_k | c_{k-1}, x_k) \\ &= \operatorname*{argmax}_{x_k} \mathcal{N}(y_k; C_{k-1} x_k, \lambda \otimes I_m) \\ &= \operatorname*{argmax}_{x_k} \log \mathcal{N}(y_k; C_{k-1} x_k, \lambda \otimes I_m) \end{aligned}$$

What is the solution? Standard derivation... Equivalent to solving,

$$x_k^* = \operatorname*{argmin}_{x_k} \|y_k - C_{k-1} x_k\|_2^2$$

Pseudoinverse.

$$x_k^* = (C_{k-1}^\top C_{k-1})^{-1} C_{k-1}^\top y_k$$

$$x_k^* = (C_{k-1}^\top C_{k-1})^{-1} C_{k-1}^\top y_k$$

matrix factorisation with linear filters

estimation

What if we try to be principled?

$$\begin{aligned} x_k^* &= \operatorname*{argmax}_{x_k} p(y_k | y_{1:k-1}, x_k) \\ &= \operatorname*{argmax}_{x_k} \int p(y_k | c, x_k) p(c | y_{1:k-1}) \mathrm{d}c \end{aligned}$$

It is not that principled at all, because the distribution $p(c|y_{1:k-1})$ can not be exact (depends on x_{k-1}^*).

Good news: You can compute this integral analytically[2].

$$p(y_k | y_{1:k-1}, x_k) = \mathcal{N}(y_k; C_{k-1} x_k, (\lambda + x_k^\top V_{k-1} x_k) \otimes I_m)$$

Bad news: Resulting optimisation problem is horrible – no solution. "Guesstimation" is much much faster.

[2] Pattern Recognition and Machine Learning, Chris Bishop, 2006.

Let us now derive the inference algorithm.

matrix factorisation with linear filters

remember the model and compare it to the generic one

The generic model that we give the filtering recursions for:

$$p(c) = \mathcal{N}(c; c_0, P_0)$$
$$p(y_k|c) = \mathcal{N}(y_k; H_k c, R_k)$$

and our model,

$$p(c) = \mathcal{N}(c; c_0, V_0 \otimes I_m)$$
$$p(y_k|c, x_k) = \mathcal{N}(y_k; C x_k, \lambda \otimes I_m)$$

Notice that we have to play and convert Cx_k into the $H_k c$ form where $c = \mathrm{vec}(C)$.

We'll fix $x_k = x_k^*$ with the "guesstimation" procedure, so recursions are same.

matrix factorisation with linear filters

remember the model and compare it to the generic one

Remember the identity we gave?

$$Ax = \mathrm{vec}(Ax) = (x^\top \otimes I_m)\mathrm{vec}(A).$$

so we have

$$Cx_k = \mathrm{vec}(Cx_k) = (x_k^\top \otimes I_m)\mathrm{vec}(C) = (x_k^\top \otimes I_m)c$$

so we obtain $H_k = (x_k^\top \otimes I_m)$. Our reformulated model,

$$p(c) = \mathcal{N}(c; c_0, V_0 \otimes I_m)$$
$$p(y_k|c, x_k) = \mathcal{N}(y_k; (x_k^\top \otimes I_m)c, \lambda \otimes I_m)$$

which exactly fits to the generic one with $R_k = \lambda \otimes I_m$, $P_0 = V_0 \otimes I_m$, and $H_k = x_k^\top \otimes I_m$.

The idea is to rewrite the following updates.

$$
\begin{aligned}
c_k &= c_{k-1} + P_{k-1} H_k^\top (H_k P_{k-1} H_k^\top + R_k)^{-1} (y_k - H_k c_{k-1}), \\
P_k &= P_{k-1} - P_{k-1} H_k^\top (H_k P_{k-1} H_k^\top + R_k)^{-1} H_k P_{k-1}.
\end{aligned}
$$

Remember: $H_k = (x_k^\top \otimes I_m)$, and $R_k = \lambda \otimes I_m$. We'll show:

- Update for c_k can be rewritten for C_k in a very efficient way.
- Update for P_k is unnecessary since it is of the following form for all k: $P_k = V_k \otimes I_m$.
  - So it suffices to update V_k which is $r \times r$ matrix (very easy).
  - Notice the gain: Instead of updating $mr \times mr$ matrix, we'll update $r \times r$!

All in all, we will derive a matrix-variate filtering algorithm!

Proposition 1. The posterior mean

$$c_k = c_{k-1} + P_{k-1}H_k^\top (H_k P_{k-1} H_k^\top + R_k)^{-1}(y_k - H_k c_{k-1})$$

can be rewritten as,

$$C_k = C_{k-1} + \frac{(y_k - C_{k-1}x_k)x_k^\top V_{k-1}^\top}{x_k^\top V_{k-1} x_k + \lambda}. \tag{15}$$

$$C_k = C_{k-1} + \frac{(y_k - C_{k-1} x_k) x_k^\top V_{k-1}^\top}{x_k^\top V_{k-1} x_k + \lambda}.$$

Proof. We write,

$$c_k = c_{k-1} + P_{k-1} H_k^\top (H_k P_{k-1} H_k^\top + R_k)^{-1} (y_k - H_k c_{k-1}),$$

and put $P_{k-1} = V_{k-1} \otimes I_m$ (see the next Prop. to see this form holds for all k) and $H_k = x_k^\top \otimes I_m$ and $R_k = \lambda \otimes I_m$, and arrive,

$$\begin{aligned} c_k = c_{k-1} &+ (V_{k-1} \otimes I_m)(x_k \otimes I_m) \\ &\Big((x_k^\top \otimes I_m)(V_{k-1} \otimes I_m)(x_k \otimes I_m) + \lambda \otimes I_m \Big)^{-1} \times \\ &(y_k - (x_k^\top \otimes I_m) c_{k-1}). \end{aligned}$$

Using the mixed product property (13) three times, using (14), and finally using (12) for the last term, one can arrive,

$$c_k = c_{k-1} + \underbrace{\left[\frac{V_{k-1} x_k}{x_k^\top V_{k-1} x_k + \lambda} \otimes I_m \right] (y_k - C_{k-1} x_k)}_{\text{Use (11) and reshape.}} . \quad \blacksquare$$

Proposition 2. The posterior covariance

$$P_k = P_{k-1} - P_{k-1} H_k^\top (H_k P_{k-1} H_k^\top + R_k)^{-1} H_k P_{k-1}$$

can be rewritten as,

$$P_k = \underbrace{\left(V_{k-1} - \frac{V_{k-1} x_k x_k^\top V_{k-1}}{x_k^\top V_{k-1} x_k + \lambda} \right)}_{V_k} \otimes I_m. \tag{17}$$

Proof. This proof is more involved. See the arXiv report.

What we perform in practice:

$$V_k = V_{k-1} - \frac{V_{k-1} x_k x_k^\top V_{k-1}}{x_k^\top V_{k-1} x_k + \lambda} \tag{18}$$

$$V_k = V_{k-1} - \frac{V_{k-1} x_k x_k^\top V_{k-1}}{x_k^\top V_{k-1} x_k + \lambda}$$

Let us run the algorithm!

$$\underbrace{\begin{bmatrix} \times & \boldsymbol{\times} & \times & \times & \times \\ \times & \boldsymbol{\times} & \times & \times & \times \\ \times & \underbrace{\boldsymbol{\times}}_{y_1} & \times & \times & \times \end{bmatrix}}_{Y} \approx \begin{bmatrix} \times & \times \\ \times & \times \\ \times & \times \end{bmatrix} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} \times & \boldsymbol{\times} & \times & \times & \times \\ \times & \boldsymbol{\times} & \times & \times & \times \\ \times & \underbrace{\boldsymbol{\times}}_{y_1} & \times & \times & \times \end{bmatrix}}_{Y} \approx \underbrace{\begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}}_{C_1} \underbrace{\begin{bmatrix} \times & \boldsymbol{\times} & \times & \times & \times \\ \times & \underbrace{\boldsymbol{\times}}_{x_1} & \times & \times & \times \end{bmatrix}}_{X}$$

$$x_1 = (C_0^\top C_0)^{-1} C_0^\top y_1$$

$$\underbrace{\begin{bmatrix} \times & \boldsymbol{\times} & \times & \times & \times \\ \times & \boldsymbol{\times} & \times & \times & \times \\ \times & \underbrace{\boldsymbol{\times}}_{y_1} & \times & \times & \times \end{bmatrix}}_{Y} \approx \underbrace{\begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}}_{C_1} \underbrace{\begin{bmatrix} \times & \boldsymbol{\times} & \times & \times & \times \\ \times & \underbrace{\boldsymbol{\times}}_{x_1} & \times & \times & \times \end{bmatrix}}_{X}$$

$$x_1 = (C_0^\top C_0)^{-1} C_0^\top y_1$$

$$C_1 = C_0 + \frac{(y_1 - C_0 x_1) x_1^\top V_0}{\lambda + x_1^\top V_0 x_1}$$

$$V_1 = V_0 - \frac{V_0 x_1 x_1^\top V_0}{x_1^\top V_0 x_1 + \lambda}.$$

$$\underbrace{\begin{bmatrix} \begin{matrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{matrix} & \underbrace{\begin{matrix} \boldsymbol{\times} \\ \boldsymbol{\times} \\ \boldsymbol{\times} \end{matrix}}_{y_2} \end{bmatrix}}_{Y} \approx \begin{bmatrix} \times & \times \\ \times & \times \\ \times & \times \end{bmatrix} \underbrace{\begin{bmatrix} \begin{matrix} \times & \times & \times & \times \\ \times & \times & \times & \times \end{matrix} & \underbrace{\begin{matrix} \boldsymbol{\times} \\ \boldsymbol{\times} \end{matrix}}_{x_2} \end{bmatrix}}_{X}$$

$$x_2 = (C_1^\top C_1)^{-1} C_1^\top y_2$$

$$\underbrace{\begin{bmatrix} \times & \times & \times & \times & \boldsymbol{\times} \\ \times & \times & \times & \times & \boldsymbol{\times} \\ \times & \times & \times & \times & \underbrace{\boldsymbol{\times}}_{y_2} \end{bmatrix}}_{Y} \approx \underbrace{\begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}}_{C_2} \underbrace{\begin{bmatrix} \times & \times & \times & \times & \boldsymbol{\times} \\ \times & \times & \times & \times & \underbrace{\boldsymbol{\times}}_{x_2} \end{bmatrix}}_{X}$$

$$x_2 = (C_1^\top C_1)^{-1} C_1^\top y_2$$

$$C_2 = C_1 + \frac{(y_2 - C_1 x_2) x_2^\top V_1}{\lambda + x_2^\top V_1 x_2}$$

$$V_2 = V_1 - \frac{V_1 x_2 x_2^\top V_1}{x_2^\top V_1 x_2 + \lambda}.$$

$$\underbrace{\begin{bmatrix} \times & \times & \times & \boldsymbol{\times} & \times \\ \times & \times & \times & \boldsymbol{\times} & \times \\ \times & \times & \times & \underbrace{\boldsymbol{\times}}_{y_k} & \times \end{bmatrix}}_{Y} \approx \underbrace{\begin{bmatrix} \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \\ \boldsymbol{\times} & \boldsymbol{\times} \end{bmatrix}}_{C_k} \underbrace{\begin{bmatrix} \times & \times & \times & \boldsymbol{\times} & \times \\ \times & \times & \times & \underbrace{\boldsymbol{\times}}_{x_k} & \times \end{bmatrix}}_{X}$$

$$x_k = (C_{k-1}^\top C_{k-1})^{-1} C_{k-1}^\top y_k$$

$$C_k = C_{k-1} + \frac{(y_k - C_{k-1} x_k) x_k^\top V_{k-1}}{\lambda + x_k^\top V_{k-1} x_k}$$

$$V_k = V_{k-1} - \frac{V_{k-1} x_k x_k^\top V_{k-1}}{x_k^\top V_{k-1} x_k + \lambda}.$$

the last extension: handling missing data

When we have missing data,

$$\begin{bmatrix} \times & & \times & & \times \\ \times & \times & & \times & \\ \times & \times & \times & & \times \end{bmatrix}$$

we model it as mask times full data,

$$\underbrace{\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}}_{M} \odot \underbrace{\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix}}_{Y}$$

and it is possible to extend our update rules to this case by putting these masks into the model. We won't go into details, and instead refer to a previous study[3].

[3]"Online Matrix Factorization via Broyden Updates", O. D. Akyildiz, `arXiv:1506.04389`.

experimental results

experiment on the olivetti dataset

We experiment on a face dataset consists of 400 faces. Each face is converted to a vector, and added to the dataset.

$$\begin{bmatrix} \times & \times \\ \times & \times \end{bmatrix} \overbrace{\Longrightarrow}^{\text{vectorisation}} \begin{bmatrix} \times \\ \times \\ \times \\ \times \end{bmatrix}$$

$$\begin{bmatrix} \times \\ \times \\ \times \\ \times \end{bmatrix}$$

$$\begin{bmatrix} \times & \times \\ \times & \times \end{bmatrix} \overbrace{\Longrightarrow}^{\text{vectorisation}} \begin{bmatrix} \times \\ \times \\ \times \\ \times \end{bmatrix}$$

$$\begin{bmatrix} \times & \times \\ \times & \times \\ \times & \times \\ \times & \times \end{bmatrix}$$

$$\begin{bmatrix}\times & \times\\ \times & \times\end{bmatrix} \overbrace{\Longrightarrow}^{\text{vectorisation}} \begin{bmatrix}\times\\ \times\\ \times\\ \times\end{bmatrix}$$

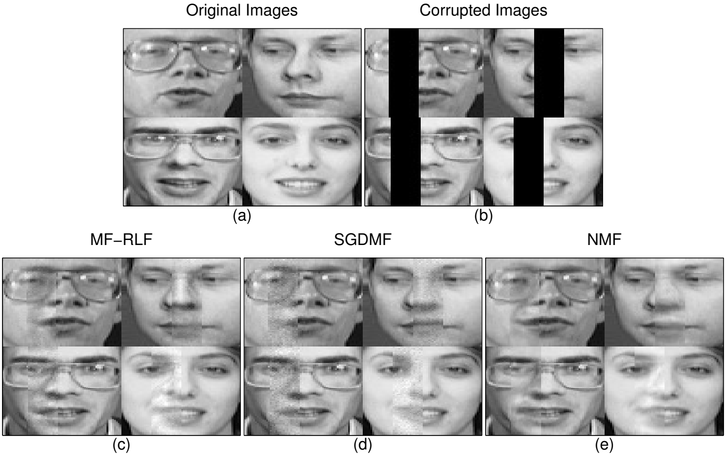
$$\begin{bmatrix}\times & \times & \times\\ \times & \times & \times\\ \times & \times & \times\\ \times & \times & \times\end{bmatrix}$$

$$\begin{bmatrix} \times & \times \\ \times & \times \end{bmatrix} \overbrace{\Longrightarrow}^{\text{vectorisation}} \begin{bmatrix} \times \\ \times \\ \times \\ \times \end{bmatrix}$$

$$\begin{bmatrix} \times & \times & \times & \cdots \\ \times & \times & \times & \cdots \\ \times & \times & \times & \cdots \\ \times & \times & \times & \cdots \end{bmatrix}$$

application to image restoration

experiment on the olivetti dataset



NMF: 1000 batch iterations. SGDMF and MF-RLF: 10 recursive passes over the dataset.

application to image restoration

signal-to-noise ratios

Initial SNR: 0.68 dB.

And the results.

Algorithm	SNR
MF-RLF	12.38 dB
SGDMF	11.75 dB
NMF	12.35 dB

some afterthoughts

- The algorithm is related to stochastic gradient descent (SGD) in a nontrivial way.
- We can say that covariance terms in the update rules "stabilize" the gradient updates of SGD.
- There is much to do: putting models on x_k, or elaborate relations to other schemes.
- Finding nice applications to time-series clustering, classification etc.

Thank you! Any questions?

—a backup slide—

Stochastic gradient descent

Let us fix X for the sake of presentation. Consider the following probability model,

$$p(Y|C,X) = \prod_{k=1}^{n} p(y_k|C,x_k)$$

and

$$p(y_k|C,x_k) = \mathcal{N}(y_k; Cx_k, I_m)$$

You could be more creative about the covariance. But let's put it that way... Applying SGD to optimisation of negative log-likelihood $-\log p(Y|C,X)$ with respect to C gives us the following update rule for C,

$$C_k = C_{k-1} + \gamma_n (y_k - C_{k-1}x_k)x_k^\top$$

Resembles what we have...

$$C_k = C_{k-1} + \frac{(y_k - C_{k-1}x_k)x_k^\top V_{k-1}}{\lambda + x_k^\top V_{k-1} x_k}$$