Noname manuscript No.
(will be inserted by the editor)

Dictionary filtering

A probabilistic approach to online matrix factorisation

Ömer Deniz Akyildiz · Joaquín Míguez

Received: date / Accepted: date

Abstract This paper investigates a link between matrix factorisation algorithms and recursive linear filters. In particular, we describe a probabilistic model in which sequential inference naturally leads to a matrix factorisation procedure. Using this probabilistic model, we derive a matrix-variate recursive linear filter that can be run efficiently in high dimensional settings and leads to the factorisation of the data matrix into a dictionary matrix and a coefficient matrix. The resulting algorithm, referred to as the *dictionary filter*, is inherently online and has easy-to-tune parameters. We provide an extension of the proposed method for the cases where the dataset of interest is time-varying and nonstationary, thereby showing the adaptability of the proposed framework to non-standard problem settings. Numerical results, which are provided for image restoration and video modelling problems, demonstrate that the proposed method is a viable alternative to existing methods.

Keywords Online matrix factorisation · Kalman filtering · stochastic optimisation

Authors acknowledge the support of the Office of Naval Research Global (award no. N62909- 15-1-2011) and *Ministerio de Economía y Competitividad* of Spain (project TEC2015-69868-C2-1-R ADVENTURE) and the regional government of Madrid (program CASICAM-CM S2013/ICE-2845).

Ömer Deniz Akyildiz
Department of Signal Theory and Communications
Universidad Carlos III de Madrid
28911, Leganés, Madrid, Spain.
E-mail: omerdeniz.akyildiz@uc3m.es

Joaquín Míguez
Department of Signal Theory and Communications
Universidad Carlos III de Madrid
28911, Leganés, Madrid, Spain.
E-mail: joaquin.miguez@uc3m.es

1 Introduction

Matrix factorisation (MF) algorithms are a cornerstone of modern signal processing, machine learning, and, more generally, computational linear algebra. Formally, we are interested in solving the problem of factorising a data matrix $Y \in \mathbb{R}^{m\times n}$ as

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

where $C \in \mathbb{R}^{m\times r}$ is the *dictionary matrix*, the columns of $X \in \mathbb{R}^{r\times n}$ are *coefficients*, and r is the *approximation rank*. In this paper we assume all matrices are real-valued. We are interested in computing both C and X recursively, or *online*, using a single (column) data vector at each time to update the factors.

Matrix factorisation methods became popular with the work on nonnegative matrix factorisation (NMF) of [1]. The authors considered the factorisation of a nonnegative data matrix $Y \in \mathbb{R}_+^{m\times n}$ into nonnegative factors $C \in \mathbb{R}_+^{m\times r}$ and $X \in \mathbb{R}_+^{r\times n}$ using a multiplicative gradient descent method (see [2] for a convergence proof). The algorithm has received significant attention due to its ability to learn important and interpretable features in an unsupervised way. Following [1], similar algorithms were also proposed for real-valued matrices and factors, as in our formulation (1), especially when the primary interest is the prediction of entries but not necessarily obtaining interpretable features. Optimisation-based approaches became popular in that avenue, i.e., taking a cost function of the form $d(Y, CX)$ and minimising it with respect to C and X using optimisation algorithms such as projected gradient descent for NMF [3] or alternating least-squares for real MF [4]. There has been a seemingly inexhaustible research activity in this area and a full literature review is out of scope for this article.

Similar to the optimisation based ideas, probabilistic approaches to MF have received considerable attention. Compared to the optimisation based methods which try to obtain point estimates of the factors, probabilistic methods aim at capturing the posterior distribution over the factors, hence quantifying the uncertainty. In [5], authors introduced a Gaussian model for real-valued MF to estimate movie ratings where factors were assumed to have independent and identically distributed (iid) entries. Similar ideas have been proposed for nonnegative factorisations [6]. Also a probabilistic interpretation of batch NMF was introduced in [7] deriving multiplicative update rules as a variational inference scheme.

All these methods were batch techniques, meaning that they require the whole dataset to update each factor at each iteration. With the rise of big datasets, these ideas became infeasible to apply. On the optimisation side, the research focus increasingly shifted to stochastic optimisation algorithms which enabled implementations of MF for large datasets. In one of the early works, NMF has been extended to the incremental setting [8]. A canonical stochastic gradient descent (SGD) based approach for general MF can be found in [9] which can be applied entry-wise or column-wise to solve the problem in (1). Similarly, one can apply the same idea to any type of differentiable cost, such as regularised versions of the cost function proposed in [9], and obtain a MF method. The idea is extended in several ways, see e.g. [10,11]. One of the fundamental limitations of these algorithms are their step-size tuning problems, a problem which has received much attention recently, see e.g. [12–14]. Every different dataset requires a different step-size and decay rate of the step-size, usually set after conducting empirical tests.

Compared to stochastic optimisation based works, online versions of probabilistic MF have received less attention. The work in [15] followed the probabilistic interpretation of NMF given in [7] to propose a sequential Monte Carlo based NMF algorithm. However, this algorithm was only applied to low dimensional problems and its applicability to realistic settings remains unclear. In [16], the same batch NMF model of [7] was implemented with stochastic variational inference techniques in the online setting.

On the other hand, sequential inference for matrix-variate linear dynamic models independently received some interest for different applications, see, e.g. [17,18]. A different perspective, closer to our approach in this paper, was introduced in [19], where matrix-variate update rules for Hessian matrices were derived as analytic inference rules in probabilistic models. As a result, the authors of [19] obtained quasi-Newton algorithms from a probabilistic perspective. However, this work focuses on the symmetric matrices whereas in this work, our main focus is general, nonsquare matrices.

In this paper, we highlight a connection between online matrix factorisation and sequential probabilistic inference. In doing so, we propose a matrix-variate dynamic linear probabilistic model in which sequential approximate inference leads to an *online* matrix factorisation algorithm. More specifically, we derive a *matrix-variate recursive filtering* method that can be readily interpreted as an online MF technique. This probabilistic characterisation brings several advantages. First, since the proposed method is based on an explicit probabilistic model, it enables the user to naturally incorporate further prior knowledge on factors (by extending the model we put forward) or dealing with non-stationary data in a principled way by putting dynamics on the dictionary matrix (see Section 3.3). Therefore, the proposed framework makes it easier to develop application-specific models and inference procedures. Secondly, from a practical perspective, compared to other *online* methods, the inference method we propose removes the need of step-size tuning and involves only easy-to-tune parameters. Specifically, the proposed algorithm does not require any step-size parameter. Its role is played by an $r \times r$ covariance matrix that is updated automatically at each iteration. We note that this is different (and computationally much cheaper) than a second-order Hessian-based approach [20], where the Hessian matrices there would need to be of the same dimension as the vectorised dictionary matrix, which is impractical in this case. Finally, as opposed to the simulation-based probabilistic MFs such as [15, 21], which only obtain samples from the posterior of the dictionary, the proposed method obtains an analytical form of the posterior distribution in terms of a Gaussian, which enables the user to quantify the uncertainty over the dictionary or diagnose convergence.

The rest of the paper is organised as follows. We introduce the probabilistic model for the factorisation problem in Section 2. Two online algorithms are derived in Section 3. In Section 4, we compare our algorithm with some popular optimisation based methods. Some illustrative experimental results on image restoration and video modeling are presented in Section 5 and, finally, Section 6 is devoted to the conclusions.

2 Probabilistic model

Notation. Throughout the paper, I_m denotes the $m \times m$ identity matrix and $\text{vec}(\cdot)$ denotes the vectorisation operation (formally, the matrix-stack operator). Specifically, for an $m \times r$ matrix Z, $z = \text{vec}(Z)$ is an $mr \times 1$

vector constructed by stacking the columns of Z. To revert this operation, we define the inverse vectorisation operation (or the inverse matrix-stack operator) as $Z = \mathrm{vec}^{-1}_{m\times r}(z)$, where the subscript indicates the dimension of Z. Intuitively this operation can be seen as a reshaping operation. We usually denote vectors with lower-case letters and matrices with capital letters.

We recall some useful definitions and equalities below. The symbol $\otimes$ denotes the Kronecker product. In particular, let A be of dimension $m \times r$ and B be of dimension $r \times n$; then [22],

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1r}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mr}B \end{bmatrix}.$$

Also, for A, B and X of appropriate dimension, we have,

$$\mathrm{vec}(AXB) = (B^\top \otimes A)\mathrm{vec}(X). \tag{2}$$

As a particular case, for an $r \times 1$ vector x we have

$$Ax = \mathrm{vec}(Ax) = (x^\top \otimes I_m)\mathrm{vec}(A). \tag{3}$$

We also draw from properties of the Kronecker product, namely the mixed product property [22],

$$(A \otimes B)(C \otimes D) = (AC) \otimes (BD), \tag{4}$$

and the inversion property [22],

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}. \tag{5}$$

We assume an argument-wise notation for probability density functions (pdf's). Given two random vectors (r.v.'s) x and y, $p(x)$ denotes the pdf of x and $p(y)$ denotes the, possibly different, pdf of y. Similarly, $p(y|x)$ is the conditional density of y given x.

2.1 Model

Recall that $Y \in \mathbb{R}^{m\times n}$ denotes the data matrix, $C \in \mathbb{R}^{m\times r}$ is the dictionary matrix, with approximation rank r, and $X \in \mathbb{R}^{r\times n}$ is the coefficient matrix. The i-th column of the data matrix is denoted $Y(:, i)$ and we use $\{1, \dots, n\}$ to denote sets of consecutive indices.

Let us consider a random mechanism for the collection of data vectors. In particular, assume that, at time k, we sample an index i_k from the uniform probability distribution over the index set $\{1, \dots, n\}$, and use it to select the data vector $y_k = Y(:, i_k)$ (specifically note that y_k denotes the observation at time k, *not* the k-th column of Y). Similarly, the i_k-th column of X is denoted $x_k = X(:, i_k)$. We use $c = \mathrm{vec}(C)$ to denote the vector form of the dictionary, while $c_k = \mathrm{vec}(C_k)$ denotes its estimate at time k. We assume a probabilistic model linking these factors, namely,

$$p(c) = \mathcal{N}(c; c_0, V_0 \otimes I_m), \tag{6}$$

$$p(y_k|c, x_k) = \mathcal{N}(y_k; Cx_k, \lambda \otimes I_m), \tag{7}$$

where $p(c)$ is a prior pdf on the dictionary vector c, $\mathcal{N}(z; \mu, \Sigma)$ denotes the Gaussian pdf of the r.v. z with mean μ and covariance matrix Σ, V_0 is an $r \times r$ a priori covariance matrix (identical for each column of C), c_0 is an $mr \times 1$ mean vector, $\lambda > 0$ is a scale parameter and $p(y_k|c, x_k)$ is the conditional pdf (assumed Gaussian) of the data vector y_k given the dictionary C and the coefficient vector x_k. The covariance matrix V_0 encodes the prior knowledge of correlations between columns of C and λ models how informative the observations are (similar to a regularisation parameter in optimisation based approaches). In this model, x_k is a static unknown parameter, while c and y_k are random vectors.

Intuitively, model (6)–(7) implies that $y_k \approx Cx_k$ (and, hence, $Y \approx CX$), where λ controls the magnitude of the approximation error. Notice that, using the identity (3) for Cx_k, we can rewrite (7) as

$$p(y_k|c, x_k) = \mathcal{N}(y_k; (x_k^\top \otimes I_m)c, \lambda \otimes I_m) \tag{8}$$

and we express the model in terms of the vector form of the dictionary. Treating c as a latent vector with observation matrix $x_k^\top \otimes I_m$ enables us to use standard linear filtering recursions in vector form. We will show, however, that working with the matrix form of the dictionary is also possible and leads to a significant reduction in computational complexity.

3 Algorithm

We assume the coefficient vectors x_k are deterministic parameters for which we aim at computing point-estimates, whereas the dictionary C is a latent random matrix and we aim at computing its posterior probability distribution (given the data in Y). The two problems are addressed in this section.

3.1 Parameter estimation

Let us assume that C_{k-1} is an estimate of the dictionary matrix computed at time $k-1$ –by a procedure to be specified later. We propose to compute a maximum likelihood estimator of the coefficient vector x_k, given the dictionary C_{k-1} and the data y_k, namely,

$$x_k^* = \underset{x_k}{\mathrm{argmax}}\, p(y_k|c_{k-1}, x_k), \tag{9}$$

where $c_{k-1} = \text{vec}(C_{k-1})$. Since the density in (9) is Gaussian, with mean $C_{k-1}x_k$, the estimator can be easily computed and yields

$$x_k^* = (C_{k-1}^\top C_{k-1})^{-1} C_{k-1}^\top y_k. \tag{10}$$

We use this update rule for the coefficients in the experiments of Section 5.

3.2 Inference of the dictionary matrix

Let us assume that $x_k = x_k^*$ is fixed via the rule in Eq. (10) and drop it from the notation for simplicity. Model (6)–(8) can then be rewritten as

$$p(c) = \mathcal{N}(c; c_0, P_0), \tag{11}$$
$$p(y_k|c) = \mathcal{N}(y_k; H_k c, R), \tag{12}$$

where $x_k = x_k^*$ is implicit, $P_0 = V_0 \otimes I_m$ and $R = \lambda \otimes I_m$. The observation matrix for this model takes the form $H_k = x_k^{*,\top} \otimes I_m$ and hence it is assumed known. Given (11)–(12), the posterior distribution of c given the data sequence $y_{1:k}$ is Gaussian and can be computed exactly [23]. To be specific, the posterior pdf is Gaussian, $p(c|y_{1:k}) = \mathcal{N}(c; c_k, P_k)$, with mean c_k and covariance matrix P_k, and can be computed exactly using a Kalman filter, which can be described by the recursive equations [23]

$$\begin{aligned} c_k &= c_{k-1} + P_{k-1}H_k^\top (H_k P_{k-1} H_k^\top + R_k)^{-1} \\ &\quad \times (y_k - H_k c_{k-1}), \end{aligned} \tag{13}$$
$$P_k = P_{k-1} - P_{k-1}H_k^\top (H_k P_{k-1} H_k^\top + R_k)^{-1} H_k P_{k-1}. \tag{14}$$

The algorithm is initialised with c_0 and P_0 in (11).

Implementing the update rules (13)–(14) is computationally inefficient, however, when $c \in \mathbb{R}^{mr}$ is large dimensional. They require the storage of a potentially very large matrices (H_k is $m \times rm$ and P_k is $rm \times rm$) which can easily make the algorithm impractical. To circumvent this limitation, we propose an equivalent, yet computationally more efficient, pair of equations for the update of the dictionary matrix $C_k = \text{vec}^{-1}_{m\times r}(c_k)$ and the covariance matrix V_k (with dimensions $r \times r$ and initialised with the matrix V_0 in (6)). The following two propositions state the form of the update rules.

Proposition 1 *The posterior covariance matrix P_k in (14) can be written as $P_k = V_k \otimes I_m$, for $k \geq 0$, where*

$$V_k = \left(V_{k-1} - \frac{V_{k-1} x_k x_k^\top V_{k-1}}{x_k^\top V_{k-1} x_k + \lambda} \right), \qquad \text{for } k \geq 1. \tag{15}$$

and V_0 is given by the prior pdf in (6).

Proof We prove this result by induction. The model is constructed in such a way that $P_0 = V_0 \otimes I_m$ and we assume that $P_{k-1} = V_{k-1} \otimes I_m$, with the sequence $\{V_l; 1 \leq l \leq k-1\}$ computed as in (15). To obtain an expression for V_k at time k, we start substituting $H_k = x_k^\top \otimes I_m$, $R = \lambda \otimes I_m$ and $P_{k-1} = V_{k-1} \otimes I_m$ into (14), which yields

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

Applying the mixed product property (4) repeatedly in the equation above we arrive at

$$\begin{aligned} P_k &= (V_{k-1} \otimes I_m) - (V_{k-1} x_k \otimes I_m) \\ &\quad \times ((x_k^\top V_{k-1} x_k + \lambda)^{-1} \otimes I_m)(x_k^\top V_{k-1} \otimes I_m), \end{aligned}$$

where we also resorted to property (5). Applying the mixed product property (4) again leads to

$$P_k = \left(V_{k-1} - \frac{V_{k-1} x_k x_k^\top V_{k-1}}{x_k^\top V_{k-1} x_k + \lambda} \right) \otimes I_m, \tag{16}$$

where the expression between brackets matches the right-hand side of (15), hence $P_k = V_k \otimes I_m$ and the proof is complete. □

Proposition 2 *The posterior mean c_k in (13) can be rewritten, in matrix form, as*

$$C_k = C_{k-1} + \frac{(y_k - C_{k-1}x_k) x_k^\top V_{k-1}^\top}{x_k^\top V_{k-1} x_k + \lambda}, \tag{17}$$

where $C_k = \text{vec}^{-1}_{m\times r}(c_k)$ is the posterior expectation of the dictionary matrix C and the sequence $\{V_k; k \geq 0\}$ is computed as in Proposition 1.

Proof Substituting $P_{k-1} = V_{k-1} \otimes I_m$ (given by Proposition 1) $H_k = x_k^\top \otimes I_m$ and $R_k = \lambda \otimes I_m$ into (13) we obtain

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

Repeatedly using the mixed product property (4) in the equation above we arrive at

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

and applying (5), together with the mixed product property again, yields

$$c_k = c_{k-1} + \left[\frac{V_{k-1} x_k}{x_k^\top V_{k-1} x_k + \lambda} \otimes I_m \right] \times (y_k - (x_k^\top \otimes I_m) c_{k-1}). \tag{18}$$

Algorithm 1 Dictionary Filter

```
1: Select $C_0$ randomly, choose an initial covariance matrix $V_0 > 0$, and set $k = 1$.
2: repeat
3:     Pick $y_k = Y(:, i_k)$ where $i_k$ is drawn from the uniform distribution over the index set $\{1, \dots, n\}$.
4:     Update the coefficient vector, the dictionary matrix and the covariance matrix as
       $x_k = (C_{k-1}^\top C_{k-1})^{-1} C_{k-1}^\top y_k$
       $C_k = C_{k-1} + \frac{(y_k - C_{k-1}x_k)x_k^\top V_{k-1}}{\lambda + x_k^\top V_{k-1} x_k}$
       $V_k = V_{k-1} - \frac{V_{k-1}x_k x_k^\top V_{k-1}}{x_k^\top V_{k-1} x_k + \lambda}$,
       respectively.
5:     $k \leftarrow k + 1$
6: until convergence
```

If we now use (3) on the last term of the right-hand side of (18) we obtain

$$c_k = c_{k-1} + \left[\frac{V_{k-1}x_k}{x_k^\top V_{k-1} x_k + \lambda} \otimes I_m \right] (y_k - C_{k-1}x_k),$$

Since $(y_k - C_{k-1}x_k)$ and $\frac{V_{k-1}x_k}{x_k^\top V_{k-1}x_k + \lambda}$ are vectors, we can rewrite this expression as,

$$c_k = c_{k-1} + \left[\operatorname{vec}\left(\frac{V_{k-1}x_k}{x_k^\top V_{k-1} x_k + \lambda} \right) \otimes I_m \right] \times \operatorname{vec}(y_k - C_{k-1}x_k), \tag{19}$$

Finally, applying (3) to the second term of the sum in the right-hand side of Eq. (19) yields

$$c_k = c_{k-1} + \operatorname{vec}\left(\frac{(y_k - C_{k-1}x_k)x_k^\top V_{k-1}^\top}{x_k^\top V_{k-1} x_k + \lambda} \right), \tag{20}$$

Now using inverse vectorisation $\operatorname{vec}_{m\times r}^{-1}(\cdot)$, we recover the update rule (17) and conclude the proof. □

The complete procedure is outlined in Algorithm 1. We hereafter refer to this method as the dictionary filter (DF). Note that this procedure has iteration complexity $\mathcal{O}(mr^2 + r^3)$. In high-dimensional scenarios where $mr^2 > r^3$, we have $\mathcal{O}(mr^2)$.

3.3 Dynamic dictionary filter

When the dataset is a (possibly nonstationary) time series, such as in video modeling problems, the prior (6) on the matrix C can be misleading since it assumes that a single dictionary for all data points can be sufficient. In these cases, one can allow C to be time-varying – hence we obtain a state-space model

$$p(\tilde{c}_0) = \mathcal{N}(\tilde{c}_0; c_0, V_0 \otimes I_m), \tag{21}$$
$$p(\tilde{c}_k|\tilde{c}_{k-1}) = \mathcal{N}(\tilde{c}_k; \tilde{c}_{k-1}, Q \otimes I_m), \tag{22}$$
$$p(y_k|\tilde{c}_k, x_k) = \mathcal{N}(y_k; \tilde{C}_k x_k, \lambda \otimes I_m), \tag{23}$$

where Q is a $r \times r$ covariance matrix. A simple modification of Algorithm 1 is sufficient to conduct inference in this model. Given the mean-covariance estimate (C_{k-1}, V_{k-1}) at time $k-1$, one needs to compute the predictive covariance matrix,

$$\tilde{V}_k = V_{k-1} + Q,$$

and use $\tilde{V}_k$ instead of V_{k-1} in all substeps within the step 4 of the Algorithm 1. We refer to this modified version as the *dynamic dictionary filter*. Intuitively, the dynamic version allows the algorithm to adapt the dictionary when the contents evolve quickly over time, such as in a sequence of video frames. We demonstrate how this property of the dynamic dictionary filter is useful in highly nonstationary problems.

4 Links with stochastic optimisation

Our algorithm relates to the MF algorithms using stochastic optimisation based approaches. In literature, MF problems are often formulated as [9],

$$\min_{C,X} \|Y - CX\|_F^2 := \sum_{k=1}^n \|y_k - Cx_k\|_2^2.$$

Let us assume that x_k is set as in Algorithm 1 given each y_k. Then, the SGD implementation for estimating C becomes update rule

$$C_k = C_{k-1} + \gamma_k (y_k - C_{k-1}x_k) x_k^\top, \tag{24}$$

where the positive step-size γ_k must be tuned to achieve the best convergence rate. In particular, it must satisfy, $\sum_{k=1}^\infty \gamma_k = \infty$ and $\sum_{k=1}^\infty \gamma_k^2 < \infty$ in order to guarantee convergence [24]. In practice, it is usually chosen as, $\gamma_k = \alpha/k^\beta$ where $\alpha > 0$ and $0.5 < \beta < 1$. The SGDMF method has iteration complexity $\mathcal{O}(mr + r^3)$ where it reduces to $\mathcal{O}(mr)$ when $mr > r^3$.

It can be seen that the update rule (24) is related to the filtering update rule (17). In our algorithm, in contrast to SGD, we do not have a step-size parameter γ_k, instead we have the gain matrix $V_{k-1}/(\lambda + x_k^\top V_{k-1}x_k))$ which is updated at each iteration with the posterior column-covariance V_k. Note that, our approach is not identical to the second-order SGD or the full filtering recursions, which are equivalent under some assumptions [25]. Those approaches would be infeasible for our

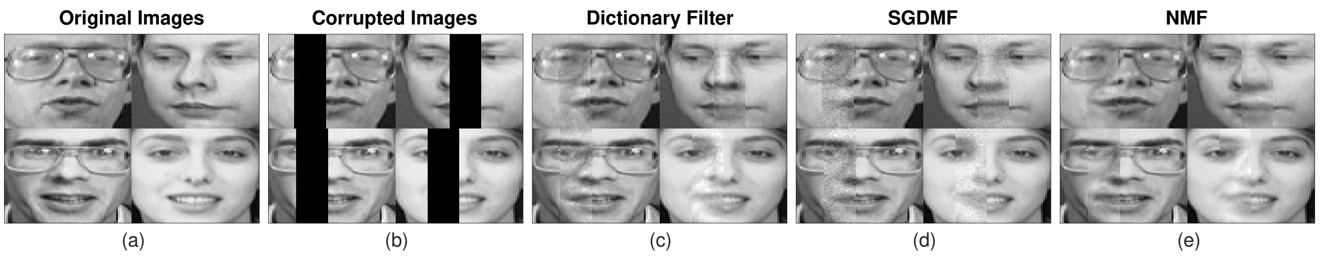

Fig. 1 Comparison of the proposed algorithm (DF) with stochastic gradient descent matrix factorisation (SGDMF), and nonnegative matrix factorisation (NMF). The NMF method was iterated 1,000 times. (a) Four original images out of the 400-member dataset. (b) Corrupted images. (c) Output of the DF algorithm. (d) Output of the SGDMF algorithm. (e) Output of the NMF algorithm. The RMSE values attained by the algorithms are 10.26 (DF), 10.13 (NMF) and 10.79 (SGDMF), starting from an initial RMSE value 70.54.

problem because the full covariance or Hessian matrices are too big to store and update compared to V_k, which captures only the posterior column covariance.

It is also possible to relate our algorithm to another class of stochastic optimisation methods, called incremental proximal methods [26]. A proximal approach to the same cost function would give the update rule (see [27] for an explicit derivation),

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

which is a special case of our algorithm, specifically if one sets $V_{k-1} = I_r$ at each iteration. This would be obviously unjustified from the probabilistic perspective. The relationship between filters and proximal algorithms is highlighted in [28].

5 Experiments

5.1 Image restoration

We tackle an image restoration problem on the Olivetti dataset [7]. This dataset consists of 400 face images of size 64×64. We vectorise each face into a column vector with dimension $m = 4096$. Since there are 400 faces, $n = 400$. We chose an approximation rank $r = 40$ and $\lambda = 2$. The factors C_0 and X_0 are initialised randomly, without imposing any structure. We choose $V_0 = I_m$ for this particular dataset (other choices do not seem to lead to better performance). It is up to the user to encode any prior knowledge about the dictionary using the covariance matrix V_0.

We deal with missing data in the images by putting masks into the model and extending the update rules for the missing data case. First, the inference step can be extended easily. We define a mask M for the whole dataset Y and denote the mask associated with y_k as m_k, more precisely $m_k = M(:, i_k)$ as $y_k = Y(:, i_k)$. Note that M is known, i.e., we know the positions of missing entries. In Algorithm 1, for the update of C_k (inference step), we simply replace the term $(y_k - C_{k-1}x_k)$ by $m_k \odot (y_k - C_{k-1}x_k)$ where $\odot$ denotes the Hadamard (element-wise) product. This corresponds to assuming having observations of form $m_k \odot y_k$ with mean $m_k \odot (Cx_k)$. Then, in accordance, we also have to compute the maximum likelihood parameter estimator x_k^* with missing data. This corresponds to solving the least squares problem,

$$\min_{x_k} \|m_k \odot (y_k - C_{k-1}x_k)\|_2^2 \tag{25}$$

For this purpose we construct a special mask, $M_k = \underbrace{[m_k, \ldots, m_k]}_{r \text{ times}}$. The rationale behind this mask can be made explicit by observing that,

$$m_k \odot (y_k - C_{k-1}x_k) = m_k \odot y_k - (M_k \odot C_{k-1})x_k.$$

Hence solving (25) is equivalent to solving the following least squares problem

$$\min_{x_k} \|m_k \odot y_k - (M_k \odot C_{k-1})x_k\|_2^2$$

which, in turn, reduces to the solution of a problem of the form $b \approx Ax$ with $b = m_k \odot y_k$ and $A = M_k \odot C_{k-1}$. Hence the solution can be found by applying the update rule (pseudoinverse operation),

$$x_k^* = ((M_k \odot C_{k-1})^\top (M_k \odot C_{k-1}))^{-1} \times (M_k \odot C_{k-1})^\top (m_k \odot y_k),$$

in Algorithm 1 for x_k.

We compare the performance of the DF, SGDMF [9] and NMF [1] algorithms. NMF can be considered as a standard benchmark for image restoration, yet we recall that it is a batch (non-recursive or offline) method. SGD is an online procedure, the same as DF. The implementation of SGDMF is similar to ours – the C_k SGD update followed by the same update rule (10).

The results of applying the three techniques can be seen in Fig. 1, along with quantitative results (root-mean squared error (RMSE) values) in the caption. In

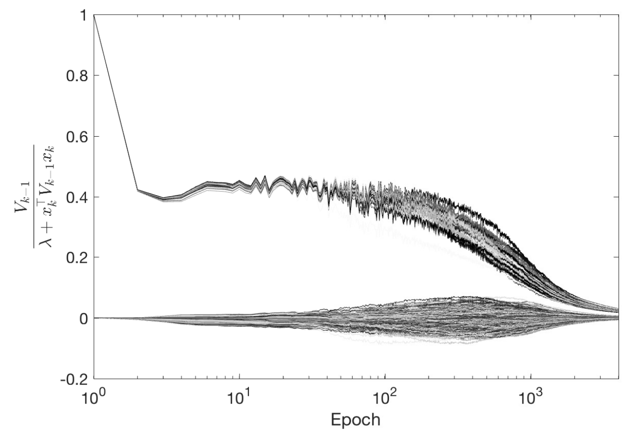

Fig. 2 Values of diagonal and upper triangular entries of $V_{k-1}/(\lambda + x_k^\top V_{k-1}x_k)$. As our update rule can be decomposed as the "step-size" and the gradient, we can see these values as step-sizes. From this perspective, it can be seen that our algorithm automatically tunes the step-sizes.

order to construct images with missing data, we randomly removed a patch consisting of %25 of all columns (for all 400 faces)[1]. The SGDMF and DF algorithms were passed 10 times over the dataset recursively (i.e., with $k = 1, \dots, 10n$). The sample images show that the proposed algorithm works well perceptually, and achieves an RMSE very close to the NMF algorithm (and better than SGDMF) with a significantly lesser computational cost.

The behaviour of the entries of the matrix $V_{k-1}/(\lambda + x_k^\top V_{k-1}x_k)$ can be seen from Fig. 2. As we initialised $V_0 = I_r$, diagonal values can be seen in the upper cluster in the plot which are decreasing from one to zero. Non-diagonal entries are initialised as zero and took nonzero values before again converging to zero. The figure hints that, as the entries go to zero, the algorithm converges empirically. This provides a natural and automated step-size tuning procedure in the form of posterior covariance matrix, which frees the user from the notorious task of step-size tuning. Note that, second-order optimisation based approaches, or the full filtering approach, would require to store an $mr \times mr$ matrix where $mr = 163,840$ in this case, which practically renders these approaches infeasible since a matrix of dimension $mr \times mr$ is not possible to fit in to the memory. In contrast, our algorithm only requires to store $r \times r$ matrix where $r = 40$, which is a lightweight computation, hence it reduces the computational burden significantly.

[1] The relative performance of the three methods remains similar when we change the percentage of missing data.

5.2 Video modeling

In this experiment, we test our algorithm on a video modeling task. The aim is to track a sequence low dimensional subspaces of the video frames and reconstruct them successfully. The video is taken from Youtube 8M dataset [29] and it has $T = 450$ frames with size 360×640 after preprocessing. We vectorised the video and obtained a dataset consisting of a 230400×450 matrix. The task is to sequentially process the video frames and obtain a sequence of dictionaries and coefficients that result in low reconstruction errors for all frames of the video. In this experiment, we have used the dynamic dictionary filter (dynamic DF) algorithm as explained in Sec. 3.3 with $Q = qI_r$ where $q = 0.05$ and $\lambda = 2$. For comparison, we have implemented SGDMF [9] and the incremental NMF (INMF) [8]. We have chosen the rank $r = 10$ for all algorithms. For SGDMF, we have chosen a constant step-size 0.1 and for INMF, we have chosen $\alpha = 0.9$. The results can be seen from Fig. 3 in terms of running times and normalized mean squared errors (NMSE). From the results, it can be concluded that dynamic DF arises as a favorable dictionary learning scheme.

6 Conclusions

We have recast the matrix factorisation problem $Y \approx CX$ as a linear filtering problem and proposed efficient matrix-variate update rules for the posterior mean and covariance of the dictionary matrix C. As a result, we have obtained an online algorithm for matrix factorisation that is flexible and computationally efficient. In particular, we have noted that the proposed method has $\mathcal{O}(mr^2)$ complexity which is independent of the number of data points. We have empirically demonstrated that the overhead of the DF compared to the SGDMF, which has $\mathcal{O}(mr)$ complexity, is small. We have also shown that the algorithm is competitive with the state-of-the-art methods in an image restoration example as well as on a video processing example which demonstrates that the proposed algorithm can successfully learn nonstationary and dynamic signals. Unlike the stochastic gradient based approaches, our algorithm does not need any parameter tuning. For future work, we plan to extend this filtering approach to nonlinear and non-Gaussian state space models where the model structure can be much richer than linear models. We believe that different methodological and application based directions, such as applications to spatio-temporal time series analysis, can be successfully pursued in the future.

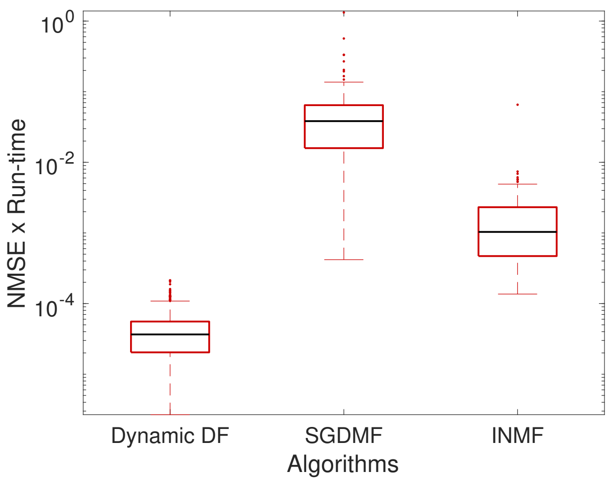

Fig. 3 The boxplot of NMSEs×run-time for 450 frames of the video for each frame. It can be seen that dynamic DF attains lower error levels compared to SGDMF and INMF. The dynamic DF takes 26.7191 seconds to run while SGDMF and INMF take 23.8624 and 20.7956 seconds respectively. While our algorithm is slightly slower, it can be seen that it is the most favorable when combined with the NMSEs.

References

1. Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, October 1999.
2. Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562, 2001.
3. Chih-Jen Lin. Projected gradient methods for non-negative matrix factorization. *Neural computation*, 19(10):2756–2779, 2007.
4. Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.
5. Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Neural Information Processing Systems (NIPS) Conference*, volume 1, pages 2–1, 2007.
6. Mikkel N Schmidt, Ole Winther, and Lars Kai Hansen. Bayesian non-negative matrix factorization. In *International Conference on Independent Component Analysis and Signal Separation*, pages 540–547. Springer, 2009.
7. Ali Taylan Cemgil. Bayesian inference for nonnegative matrix factorisation models. *Computational Intelligence and Neuroscience*, pages 4:1–4:17, January 2009.
8. Serhat S Bucak and Bilge Gunsel. Incremental subspace learning via non-negative matrix factorization. *Pattern recognition*, 42(5):788–797, 2009.
9. Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM, 2011.
10. Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *The Journal of Machine Learning Research*, 11:19–60, 2010.
11. Naiyang Guan, Dacheng Tao, Zhigang Luo, and Bo Yuan. Online nonnegative matrix factorization with robust stochastic approximation. *IEEE Transactions on Neural Networks and Learning Systems*, 23(7):1087–1099, 2012.
12. John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
13. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations*, 2015.
14. Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. In *Advances In Neural Information Processing Systems*, pages 181–189, 2015.
15. S. Yildirim, A. T. Cemgil, and S. S. Singh. An online expectation-maximisation algorithm for nonnegative matrix factorisation models. In *16th IFAC Symposium on System Identification (SYSID 2012)*, 2012.
16. John Paisley, D Blei, and Michael I Jordan. Bayesian nonnegative matrix factorization with stochastic variational inference. In *volume Handbook of Mixed Membership Models and Their Applications, chapter 11*. Chapman and Hall/CRC, 2015.
17. Carlos M Carvalho, Mike West, et al. Dynamic matrix-variate graphical models. *Bayesian analysis*, 2(1):69–97, 2007.
18. K Triantafyllopoulos. Reference priors for matrix-variate dynamic linear models. *Communications in Statistics—Theory and Methods*, 37(6):947–958, 2008.
19. Philipp Hennig and Martin Kiefel. Quasi-newton methods: A new direction. *The Journal of Machine Learning Research*, 14(1):843–865, 2013.
20. Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific, 1999.
21. Sungjin Ahn, Anoop Korattikara, Nathan Liu, Suju Rajan, and Max Welling. Large-scale distributed Bayesian matrix factorization using stochastic gradient MCMC. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 9–18. ACM, 2015.
22. David A Harville. *Matrix algebra from a statistician's perspective*, volume 1. Springer, 1997.
23. Brian DO Anderson and John B Moore. Optimal filtering. *Englewood Cliffs, NJ: Pren*, 1979.
24. Léon Bottou. Online learning and stochastic approximations, 1998.
25. Yann Ollivier. Online natural gradient as a kalman filter. *arXiv:1703.00209*, 2017.
26. Dimitri P Bertsekas. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, 2010:1–38, 2011.
27. Ömer Deniz Akyıldız. Online matrix factorization via Broyden updates. *arXiv:1506.04389*, 2015.
28. Ömer Deniz Akyıldız, Víctor Elvira, and Joaquín Míguez. The incremental proximal method: A probabilistic perspective. In *Proc. of IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 4279–4283, 2018.
29. Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *arXiv:1609.08675*, 2016.