Matrix Factorisation with Linear Filters

Ömer Deniz Akyıldız

Abstract—This text investigates relations between two well-known family of algorithms, matrix factorisations and recursive linear filters, by describing a probabilistic model in which approximate inference corresponds to a matrix factorisation algorithm. Using the probabilistic model, we derive a matrix factorisation algorithm as a recursive linear filter. More precisely, we derive a matrix-variate recursive linear filter in order to perform efficient inference in high dimensions. We also show that it is possible to interpret our algorithm as a nontrivial cost function and also the model definitions are slightly different. And finally, we apply these ideas to matrix factorisation problem. The provided probabilistic characterisation opens up many possibilities for incorporating further prior knowledge, or dealing with nonstationary data in a principled way by putting dynamics on the dictionary matrix.

In the following subsection, we’ll give some identities which will be very useful in proofs. In Section II, we describe our generative model for matrix factorisation. In Section III, we derive our algorithm as an estimation and inference algorithm in the probabilistic model described in the Section II. In Section IV, we describe the relation between SGD based matrix factorisation, and our algorithm. In Section V, we demonstrate our algorithm on an image restoration task. In Section VI, we conclude.

A. Some useful linear algebra

We will be heavily using the following identities from [10].

1. vec(AXB) = (Bᵀ ⊗ A)vec(X) (2)

A particular case where this identity will be useful for us is when dim(A) = m × n and dim(B) = r × 1. So let us note the particular case in a more useful form to us,

(xᵀ ⊗ I_m)vec(A) = vec(Ax) = Ax. (3)

where Ax is also a vector, dim(x) = r × 1, and I_m is m × m matrix. For a matrix M where dim(M) = m × r, let m = vec(M) is a nr × 1 vector. To revert this operation, we define the reshaping operator: vec⁻¹_m×r(m) = M. Kronecker products also have the following mixed product property,

(A ⊗ B)(C ⊗ D) = (AC) ⊗ (BD), (4)

and the following “inversion” property,

(A ⊗ B)⁻¹ = A⁻¹ ⊗ B⁻¹. (5)
II. The Probabilistic Model

Let \( Y \in \mathbb{R}^{m \times n} \) be the data matrix, and \( C \in \mathbb{R}^{m \times r} \) and \( X \in \mathbb{R}^{r \times n} \). Let us denote the \( i \)’th column of the data matrix \( Y \) with \( Y(:, i) \), and \( |n| = \{1, \ldots, n\} \). The observations are generated in the following way: At time \( k \), we randomly sample an index \( i_k \sim [n] \). And we set \( y_k = Y(:, i_k) \). So \( y_k \) denotes the observation at time \( k \) but not \( k \)’th column. Similarly, the associated column of \( X \) is denoted with \( x_k \), and \( x_k = X(:, i_k) \). For example if \( i_k = 2 \), then \( y_k \) would be the second column of \( Y \), and \( x_k \) would be the second column of \( X \). We denote the dictionary matrix with \( C \), and \( c = \text{vec}(C) \). This also holds for \( c_k = \text{vec}(C_k) \) where \( C_k \) is \( m \times r \) matrix stands for estimate of \( C \) at iteration \( k \).

In this work, we consider the following probabilistic model,

\[
p(c) = \mathcal{N}(c; c_0, V_0 \otimes I_m)
\]

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

Note that \( x_k \) is a static unknown model parameter vector. On the other hand, \( c \) and \( y_k \) are random vectors, and treated as such. To motivate the model, notice that using identity (3) for \( (x_k^\top \otimes I_m)c \), we can rewrite the likelihood (3) in the following form,

\[
p(y_k|c, x_k) = \mathcal{N}(y_k; Cx_k, \lambda \otimes I_m).
\]

In the matrix factorisation setup, we would like to assume \( y_k \approx Cx_k \) for each \( k \), here this corresponds to assuming Gaussian noise. Using the model (6) and (7), we would like to estimate both \( x_k \) and \( C \) given the observations \( y_{1:k} \), i.e. observations up to time \( k \).

III. Parameter Estimation and Inference

From the viewpoint of probabilistic (or Bayesian) inference, coefficients \( x_k \) are static parameters to be estimated (typically by some optimisation formulation), and in contrast, the dictionary matrix \( C \) is a latent variable that is to be inferred through its posterior distribution. In this section, we’ll show how to perform parameter estimation for coefficients, and inference for the dictionary matrix.

A. Parameter estimation: Finding coefficients

To estimate the parameters \( x_k \) associated with a given observation \( y_k \), we formulate the following maximisation problem,

\[
x_k^* = \arg\max_{x_k} p(y_k|x_{k-1}, x_k)
\]

Since this density is a Gaussian with mean \( C_{k-1}x_k \), the solution is the pseudoinverse,

\[
x_k^* = \left(C_{k-1}^\top C_{k-1}\right)^{-1}C_{k-1}^\top y_k.
\]

Note that in this work, we use this update rule in the experiments. However, just to note, a very intriguing approach would be maximising the marginal likelihood \( p(y_k|y_{1:k-1}, x_k) \) by integrating out \( c \). Unfortunately, the optimisation part is intractable and we will discuss this elsewhere.1

B. Inference: Finding the dictionary matrix

In this subsection, we assume \( x_k \) is fixed and \( x_k = x_k^* \), and we suppress \( x_k \) from the notation. We consider the model (6) and (7), and solve the posterior inference problem. We can rewrite this model in a generic way,

\[
p(c) = \mathcal{N}(c; c_0, P_0),
\]

\[
p(y_k|c) = \mathcal{N}(y_k; H_k c, R),
\]

where \( P_0 = V_0 \otimes I \) and \( R = \lambda \otimes I \). Since we fix \( x_k \) for all \( k \), we suppress the \( x_k \) from the notation, and use generic \( H_k \) observation matrix which is assumed to be known now. Given this model and fixed parameters, it is well-known that given observations up to time \( k \), the posterior distribution \( p(c|y_{1:k}) \) is Gaussian too [12]. We denote this posterior density by \( p(c|y_{1:k}) = \mathcal{N}(c; c_k, P_k) \). The mean \( c_k \) and covariance \( P_k \) can be found by a recursive least squares filter (recursive linear filter) algorithm. Given observations \( y_{1:k} \), the mean \( c_k \) is given by [12].

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

and the covariance of the posterior is given by,

\[
P_k = P_{k-1} - P_{k-1}H_k^\top (H_k P_{k-1}H_k^\top + R)\^{-1}H_k P_{k-1}.
\]

Implementing these update rules would be very inefficient as \( c \in \mathbb{R}^{mr} \) might be a very high-dimensional vector. This requires to store a huge observation matrix \( H_k \) and a huge covariance matrix \( P_k \) which can easily become an impractical problem to solve. But fortunately we can obtain a very efficient matrix-variate update rule using the following proposition.

**Proposition 1.** The posterior mean \( c_k \) which is given by,

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

can be rewritten as,

\[
C_k = C_{k-1} + \frac{(y_k - C_{k-1}x_k)x_k^\top V_k \lambda^{-1}}{x_k^\top V_k x_k + \lambda}.
\]

**Proof.** We put \( P_{k-1} = V_{k-1} \otimes I_m \) (see Prop. 2 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,

\[
c_k = c_{k-1} + (V_{k-1} \otimes I_m)(x_k \otimes I_m)
\]

\[
\left((x_k^\top \otimes I_m)(V_{k-1} \otimes I_m)x_k \otimes I_m) + \lambda \otimes I_m\right)^{-1} \times
\]

\[
(y_k - (x_k^\top \otimes I_m)c_{k-1}).
\]

Using the mixed product property (4) three times, using (5), and finally using (3) for the last term, one can arrive,

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

Use (2) and reshaping with \( \text{vec}^{-1} \), we obtain Eq. (11).
earlier work based on Broyden updates [13]. In the following proposition, we derive an efficient posterior covariance update to use in mean update [11].

Proposition 2. The posterior covariance update,

\[ P_k = P_{k-1} - P_{k-1}H_k^T (H_kP_{k-1}H_k^T + R)^{-1}H_kP_{k-1}, \]

can be rewritten as,

\[ P_k = \left(V_{k-1} - \frac{V_{k-1}x_kx_k^T V_{k-1}}{x_k^TV_{k-1}x_k + \lambda} \right) \otimes I_m. \]  \hfill (12)

Proof. We start by putting \( H_k = x_k^T \otimes I_m \) and \( R = \lambda \otimes I_m \). So we arrive,

\[ P_k = P_{k-1} - P_{k-1}(x_k \otimes I_m)(x_k^T \otimes I_m)P_{k-1}(x_k \otimes I_m) + \lambda \otimes I_m)^{-1}(x_k^T \otimes I_m)P_{k-1}. \]

We also put \( P_k = V_k \otimes I_m \). We will show that this form holds also \( P_k \), and since \( P_0 \) is also of this form, by induction, we arrive that it holds for all \( k \). Let us put,

\[ P_k = (V_{k-1} \otimes I_m) - (V_{k-1} \otimes I_m)(x_k \otimes I_m)\times ((x_k^T \otimes I_m)(V_{k-1} \otimes I_m)(x_k \otimes I_m) + \lambda \otimes I_m)^{-1}\times
\]

\[ (x_k^T \otimes I_m)(V_{k-1} \otimes I_m). \]

By using mixed product property \( \text{(4)} \) several times, we obtain,

\[ P_k = (V_{k-1} \otimes I_m) - (V_{k-1} x_k \otimes I_m)\times
\]

\[ ((x_k^T V_{k-1}x_k + \lambda)^{-1} \otimes I_m)(x_k^T V_{k-1}x_k + \lambda) \otimes I_m. \]

where we also used property \( \text{(5)} \). Few more uses of mixed product property leads to,

\[ P_k = (V_{k-1} \otimes I_m) - \frac{V_{k-1}x_kx_k^TV_{k-1}}{x_k^TV_{k-1}x_k + \lambda} \otimes I_m. \]

Thus we can say that \( P_k = V_k \otimes I_m \) where,

\[ V_k = V_{k-1} - \frac{V_{k-1}x_kx_k^TV_{k-1}}{x_k^TV_{k-1}x_k + \lambda}. \]  \hfill (13)

We give the overall algorithm in Algorithm 1. We name it as matrix factorisation based on recursive linear filter (MF-RLF).

\begin{algorithm}[!h]
\caption{MF-RLF}
\begin{algorithmic}[1]
\STATE Initialise \( C_0 \) randomly and set \( k = 1 \).
\REPEAT
\STATE Pick \( y_k = Y(\cdot; i_k) \) where \( i_k \sim [n] \) uniformly random.
\STATE Perform,
\end{algorithmic}

\[ x_k = (C_{k-1}^TC_{k-1})^{-1}C_{k-1}^Ty_k \]

\[ C_k = C_{k-1} + \frac{(y_k - C_{k-1}x_k)x_k^TV_{k-1}}{\lambda + x_k^TV_{k-1}x_k} \]

\[ V_k = V_{k-1} - \frac{V_{k-1}x_kx_k^TV_{k-1}}{x_k^TV_{k-1}x_k + \lambda}. \]

\UNTIL{convergence}
\end{algorithm}

Algorithmically, it is a simple modification to the Algorithm I. Define \( Q_k = Q_V \otimes I_m \) where \( Q_V \) is \( r \times r \) covariance matrix. So to obtain the matrix-variate Kalman filter, it suffices to perform the following step just before step 4 of the Algorithm I,

\[ V_{k|k-1} = V_{k-1} + Q_V, \]

and use \( V_{k|k-1} \) for updating \( C_k \) and \( V_k \). We think that it could be very useful to develop an explicit model when one needs a “forgetting” property in the dictionary. It can be a principled alternative to what is called “forgetting factor” of the RLS when performing matrix factorisations. \( Q_V \) can be actively used to add a dynamic to the dictionary matrix. We leave this potential application to the future work.

IV. RELATION TO STOCHASTIC GRADIENT DESCENT

Our algorithm can be interpreted as a version of stochastic gradient descent with a nontrivial and non-scalar step size. The interpretation is given as follows: Suppose \( y_k \) are iid draws conditioned on \( C \) (estimating \( X \) will be identical to previous case, so assume it is known), and we would like to maximise the following joint likelihood of the dataset,

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

and assume this likelihood is defined as

\[ p(y_k|C, x_k) = \mathcal{N}(y_k; C x_k, I). \]

Then after a bit of calculation, one can show that applying SGD to the negative log-likelihood results in the following iteration,

\[ C_k = C_{k-1} + \gamma_k(y_k - C_{k-1}x_k)x_k^T. \]  \hfill (14)

First of all, putting \( \gamma_k = 1/(\lambda + x_k^TV_{k-1}x_k) \) recovers Broyden’s rule again from a different perspective: maximum likelihood estimation via SGD \footnote{There are other ways, e.g. embedding step-size into the covariance. So this hints for an interesting connection between the step-size of the SGD and posterior covariance of the recursive linear-Gaussian models.} But note that this does not ensure that the usual assumptions on the step-size is satisfied, hence the convergence is questionable \footnote{We note that, the update rule \hfill (11) that is proposed in this paper is different than \hfill (14) as we also have a matrix \( V_k \) which can not be embedded into the step-size in a trivial way.}. We will show that this form is the gradient descent with a nontrivial and non-scalar step size. The potential application to the future work.
V. APPLICATION TO IMAGE RESTORATION

We demonstrate our algorithm on an image restoration task on the Olivetti dataset [2]. This dataset consists of 400 face images of size 64 × 64. We vectorise each face into a column vector with dimension 4096, so \( m = 4096 \) in this problem. Since there are 400 faces in the dataset, \( n = 400 \). We chose \( r = 40 \) as an approximation rank and \( \lambda = 2 \). We initialised factors randomly without imposing any structure. We choose \( V_0 = I \) for this particular dataset, other choices lead to poorer performance. But it is entirely up to user to encode a prior knowledge about dictionary by using covariance matrix \( V_0 \) that encodes a qualitative knowledge about the structure between \( r \) columns of the dictionary matrix.

We deal with missing data using exact same methodology described in [13]. So we define a mask \( M \), and denote the mask associated with \( y_k \) with \( m_k \). So in the Algorithm 1 we replace \((y_k - C_{k-1}x_k)\) term by \( m_k \odot (y_k - C_{k-1}x_k) \). Also while updating \( C_k \), we construct a special mask,

\[
M_{C_k} = [m_k, \ldots, m_k],
\]

and apply the following update,

\[
x_k = ((M_{C_{k-1}} \odot C_{k-1})^T (M_{C_{k-1}} \odot C_{k-1}))^{-1} \times (M_{C_{k-1}} \odot C_{k-1})^T (m_k \odot y_k),
\]

in the Algorithm 1 for \( x_k \). Note that all these reformulations can be derived from the model by putting masks into the model. We left them out for simplicity. For the more details of this missing data handling scheme see [13]. We use this both for SGD and MF-RLF.

We give comparisons with both SGD and NMF. We give a comparison with NMF because we think that the most basic task of an online algorithm is to compete with the state-of-the-art batch methods. In general, many online algorithms fail at fulfilling this task because datasets which one can experiment batch algorithms are too small for online learning. In this section, we show that our algorithm fulfils this hard task: It works as good as NMF – the standard batch benchmark – on image restoration. As our algorithm bears some similarities with SGD, we also give a comparison with SGD as an online algorithm. The implementation is similar to ours – the \( C_k \) update (14) subsequently followed by pseudoinverse. The visual results can be seen from Fig. 1 and SNR values are tabulated in Table I MF-RLF and SGDMF passed recursively 10 times over the dataset, i.e. using a single observation each iteration. We ran NMF with 1000 batch passes over data. This shows these recursive algorithms uses data much more efficiently.

Results show that our algorithm works well perceptually, and achieve same SNR values with NMF although it only passed 10 times over the dataset.

VI. CONCLUSION

We presented a matrix factorisation algorithm which makes use of linear filters. We recast the factorisation problem into the linear filtering problem, and propose efficient matrix-variate update rules for the Gaussian posterior summary statistics. The algorithm can trivially be extended for dynamic models on dictionary matrix where one can model changing nature of the dataset in a principled way. For the future work, we think to extend this filtering approach to nonlinear and non-Gaussian state space models where the model structure can be much more richer than linear models. Putting a nonlinear dynamics on \( x_k \) poses new challenges for sequential inference schemes in high-dimensions, and calls for Rao-Blackwellisation of state-of-the-art algorithms (such as [15]) proposed for high-dimensional filtering. Another potential use of our algorithm can be based on uncertainty estimates: Covariance uncertainty can be used to stop unnecessary computations, and save enormous time in a related fashion to probabilistic numerics [16]. We hope to pursue different methodological and application based directions for the future.

ACKNOWLEDGEMENTS

I am grateful to Philipp Hennig for very valuable discussions. I am thankful to A. Taylan Cemgil for his support and suggestions. I am thankful to Baris Eвrim Demiroz and Thomas Schön for discussions. This work is supported by TÜBİTAK under the grant number 113M492 (PAVERA).
REFERENCES

[1] 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.

[2] A. T. Cemgil, “Bayesian inference in non-negative matrix factorisation models,” *Computational Intelligence and Neuroscience*, no. Article ID 785152, 2009.

[3] J. Z. Sun, K. R. Varshney, and K. Subbian, “Dynamic matrix factorization: A state space approach,” in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1897–1900.

[4] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. 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*. ACM, 2011, pp. 69–77.

[5] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, “Online learning for matrix factorization and sparse coding,” *The Journal of Machine Learning Research*, vol. 11, pp. 19–60, 2010.

[6] K. Skretting and K. Engan, “Recursive least squares dictionary learning algorithm,” *Signal Processing, IEEE Transactions on*, vol. 58, no. 4, pp. 2121–2130, 2010.

[7] Y. Zhang, H. Wang, and W. Wang, “An analysis dictionary learning algorithm based on recursive least squares,” in *Signal Processing (ICSP), 2014 12th International Conference on*. IEEE, 2014, pp. 831–835.

[8] P. Hennig and M. Kiefel, “Quasi-newton methods: A new direction,” *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 843–865, 2013.

[9] P. Hennig, “Probabilistic interpretation of linear solvers,” *SIAM Journal on Optimization*, vol. 25, no. 1, pp. 234–260, 2015.

[10] D. A. Harville, *Matrix algebra from a statistician’s perspective*. Springer, 1997, vol. 1.

[11] K. B. Petersen, M. S. Pedersen et al., “The matrix cookbook,” *Technical University of Denmark*, vol. 7, p. 15, 2008.

[12] S. Särkkä, *Bayesian filtering and smoothing*. Cambridge University Press, 2013, no. 3.

[13] Ö. D. Akyıldız, “Online matrix factorization via broyden updates,” *arXiv preprint, arXiv:1506.04389*, 2015.

[14] L. Bottou, “Online learning and stochastic approximations,” 1998.

[15] C. A. Naesseth, F. Lindsten, and T. B. Schön, “Nested sequential monte carlo methods,” in *Proceedings of the 32nd International Conference on Machine Learning, Lille, France*, 2015.

[16] P. Hennig, M. A. Osborne, and M. Girolami, “Probabilistic numerics and uncertainty in computations,” in *Proc. R. Soc. A*, vol. 471, no. 2179. The Royal Society, 2015, p. 20150142.