Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Incremental Stochastic Factorization for Online Reinforcement Learning

André M. S. Barreto and **Rafael L. Beirigo**
Laboratório Nacional de Computação Científica
Petrópolis, RJ, Brazil
{amsb, rafaelb}@lncc.br

Joelle Pineau and **Doina Precup**
School of Computer Science, McGill University
Montreal, QC, Canada
{jpineau, dprecup}@cs.mcgill.ca

Abstract

A construct that has been receiving attention recently in reinforcement learning is stochastic factorization (SF), a particular case of non-negative factorization (NMF) in which the matrices involved are stochastic. The idea is to use SF to approximate the transition matrices of a Markov decision process (MDP). This is useful for two reasons. First, learning the factors of the SF instead of the transition matrices can reduce significantly the number of parameters to be estimated. Second, it has been shown that SF can be used to reduce the number of operations needed to compute an MDP's value function. Recently, an algorithm called expectation-maximization SF (EMSF) has been proposed to compute a SF directly from transitions sampled from an MDP. In this paper we take a closer look at EMSF. First, by exploiting the assumptions underlying the algorithm, we show that it is possible to reduce it to simple multiplicative update rules similar to the ones that helped popularize NMF. Second, we analyze the optimization process underlying EMSF and find that it minimizes a modified version of the Kullback-Leibler divergence that is particularly well-suited for learning a SF from data sampled from an arbitrary distribution. Third, we build on this improved understanding of EMSF to draw an interesting connection with NMF and probabilistic latent semantic analysis. We also exploit the simplified update rules to introduce a new version of EMSF that generalizes and significantly improves its precursor. This new algorithm provides a practical mechanism to control the trade-off between memory usage and computing time, essentially freeing the space complexity of EMSF from its dependency on the number of sample transitions. The algorithm can also compute its approximation incrementally, which makes it possible to use it concomitantly with the collection of data. This feature makes the new version of EMSF particularly suitable for online reinforcement learning. Empirical results support the utility of the proposed algorithm.

1 Introduction

It is widely recognized that efficient learning is only possible if one exploits some kind of regularity in the problem (Györfi et al. 2002). In reinforcement learning, a structural regularity that has been receiving attention recently is a particular case of non-negative factorization (NMF) called *stochastic factorization* (SF). As the name suggests, in this case a stochastic matrix $\mathbf{P} \in \mathbb{R}^{n \times n}$ is represented as the product of two stochastic matrices $\mathbf{D} \in \mathbb{R}^{n \times m}$ and $\mathbf{K} \in \mathbb{R}^{m \times n}$, with $m < n$.

In the context of reinforcement learning, one is interested in Markov decision processes (MDPs) whose transition matrices can be well approximated by SFs. This interest was originally triggered by an intriguing property of SF: if we *swap* the factors of the factorization, we obtain another stochastic matrix $\bar{\mathbf{P}} = \mathbf{KD}$ that retains some important properties of the Markov chain represented by $\mathbf{P}$ (Barreto and Fragoso 2011). By replacing the transition matrices of an MDP with their reduced counterparts, one can derive a smaller MDP whose value function can be used to recover the corresponding function of the original model (Barreto, Pineau, and Precup 2014). This strategy of replacing $\mathbf{P}$ with $\bar{\mathbf{P}}$ is sometimes referred to as the "SF trick."

One difficulty in applying the SF trick in practice is the need to compute the factorization itself, which has been shown to be an NP-hard problem (Vavasis 2009). In a recent paper we have suggested looking at the problem from a different perspective: instead of assuming one knows the transition matrix $\mathbf{P}$ and wants to compute the factors $\mathbf{D}$ and $\mathbf{K}$, we consider the scenario where one only has access to transitions sampled from $\mathbf{P}$ (Barreto et al. 2015). In this case one could of course compute an estimate of $\mathbf{P}$ by counting the occurrences of transitions, which would take us back to the original scenario. Instead, we proposed an approach to compute $\mathbf{D}$ and $\mathbf{K}$ directly from data, without ever computing an explicit estimate of $\mathbf{P}$. One obvious advantage of this strategy is that the number of parameters to be estimated is reduced from n^2 to $2nm$; since m is a parameter of the algorithm, it serves as a practical device to control the trade off between the estimation and approximation errors in the model $\mathbf{DK} \approx \mathbf{P}$ (Györfi et al. 2002). Thus, besides the SF trick, the possibility of using SF to speed up learning further motivates our interest on this model.

The algorithm to compute an SF from data is an expectation-maximization (EM) method called *expectation-maximization stochastic factorization* (EMSF) (Barreto et al. 2015). In this paper we take a closer look at EMSF. First, by exploiting the assumptions underlying the algorithm, we show that it is possible to reduce its original update rules to simple multiplicative rules similar to the ones that helped popularize non-negative factorization (NMF) (Lee

and Seung 1999; 2001). Second, we analyze the optimization problem underlying EMSF and find that the dissimilarity function implicitly minimized by it is a specialization of the Kullback-Leibler divergence which is particularly well-suited for the scenario where the factorization must be learned from data sampled from an arbitrary distribution. Third, we build on this improved understanding of EMSF to draw a connection with probabilistic latent semantic analysis (PLSA), which in turn unveils a link with known NMF algorithms (Hofmann 1999).

Besides highlighting the connection between EMSF and other algorithms from the literature, the simplified update rules also shed light on the mechanics of the algorithm. We exploit this to develop an incremental version of EMSF that generalizes and improves its precursor. By tuning the parameters of the new algorithm one can set the right balance between memory usage and computing time. In the slowest end of the spectrum we recover the original EMSF; at the other extreme we encounter an algorithm that is considerably faster than its precursor. The incremental version of EMSF can also be applied in parallel with the collection of data, which makes it particularly suitable to online reinforcement learning (Sutton and Barto 1998). We test the new algorithm empirically on simple problems.

2 Background and Notation

Random variables are represented by capital letters; we use the notation $A_{1:\tau}$ to refer to a sequence of variables $A_1, A_2, ..., A_\tau$. Scalar variables are represented by small letters; boldface capital letters and boldface small letters are used to denote matrices and vectors, respectively. Given a matrix $\mathbf{A}$, $\mathbf{A}_i$ is its i^{th} row, $\mathbf{A}^j$ is its j^{th} column, and $\mathbf{A}_{ij}$ is the element in the i^{th} row and j^{th} column.

A *stochastic matrix* has only non-negative elements and its rows sum to 1. We call a square stochastic matrix a *transition matrix*. As described in the introduction, SF is a relation $\mathbf{P} = \mathbf{DK}$ in which $\mathbf{P} \in \mathbb{R}^{n\times n}$, $\mathbf{D} \in \mathbb{R}^{n\times m}$, and $\mathbf{K} \in \mathbb{R}^{m\times n}$ are stochastic. The SF of a transition matrix makes it possible to apply the SF trick: by *swapping* the factors of the factorization one gets another transition matrix $\bar{\mathbf{P}} = \mathbf{KD}$, potentially much smaller than the original, that retains some fundamental properties of $\mathbf{P}$, such as the number of recurrent classes and their respective periods (see Barreto and Fragoso's paper for details, 2011).

Here we are interested in the application of the SF trick to MDPs. We assume the reader is familiar with the basics of reinforcement learning and MDPs; for our purposes it suffices to see the latter as a collection of u transition matrices $\mathbf{P}^a \in \mathbb{R}^{n\times n}$, each one describing the dynamics associated with an *action* a (Puterman 1994). Let M be an MDP with transition matrices $\mathbf{P}^a$. Roughly speaking, we want to compute approximations $\mathbf{D}^a\mathbf{K}^a \approx \mathbf{P}^a$, define a new MDP $\bar{M}$ whose transition matrices are $\bar{\mathbf{P}}^a = \mathbf{K}^a\mathbf{D}^a$, solve $\bar{M}$, and then use the solution to compute an approximation of the value function of M (Barreto, Pineau, and Precup 2014).[1]

[1]The astute reader will notice a similarity between this strategy and Bertsekas's (2011) aggregation/disaggregation scheme. Note though that in the latter the dynamics of the reduced model is given by $\mathbf{KPD}$ rather than $\mathbf{KD}$, and the multiplication $\mathbf{DK}$ is *not* seen as an approximation of $\mathbf{P}$.

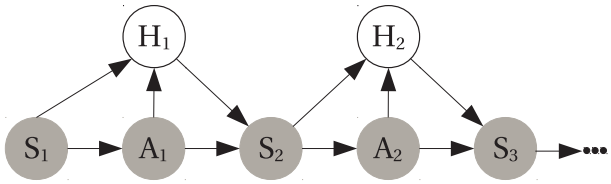

Figure 1: Graphical model of an SFM

We have recently proposed an algorithm to compute $\mathbf{D}^a\mathbf{K}^a \approx \mathbf{P}^a$ using only transitions sampled from an MDP M (Barreto et al. 2015). In order to do so, we endowed stochastic factorization with a probabilistic interpretation, which we now briefly describe. Let $\mathcal{S} \equiv \{1, 2, ..., n\}$, $\mathcal{H} \equiv \{1, 2, ..., m\}$, and $\mathcal{A} \equiv \{1, 2, ..., u\}$. Consider the stochastic process $(S_1, A_1, H_1, S_2, A_2, H_2, ...)$, where $S_t \in \mathcal{S}$ are observable states, $A_t \in \mathcal{A}$ are observable actions, and $H_t \in \mathcal{H}$ are hidden states. The proposed probabilistic model builds on the following Markov assumptions:

(i) $\Pr(S_t|H_{t-1}, A_{t-1}, ..., S_1) = \Pr(S_t|H_{t-1}, A_{t-1})$;

(ii) $\Pr(H_t|A_t, S_t, H_{t-1}, ..., S_1) = \Pr(H_t|A_t, S_t)$;

(iii) $\Pr(A_t|S_t, H_{t-1}, A_{t-1}, ..., S_1) = \Pr(A_t|S_t)$.

The assumptions above are depicted as a graphical model in Figure 1. Given $\tau > 0$, let $W_{1:\tau} \equiv \{S_1, A_1, ..., A_{\tau-1}, H_{\tau-1}, S_\tau\}$. It is easy to compute the probability of any possible instantiation of $W_{1:\tau}$ under Assumptions (i), (ii), and (iii):

$$\Pr(W_{1:\tau}) = \Pr(S_1)\prod_{t=1}^{\tau-1}\Pr(S_{t+1}|H_t, A_t)\Pr(A_t|S_t)\Pr(H_t|S_t, A_t).$$

Thus, the only quantities needed to fully characterize a stochastic process with the properties above are $\Pr(S_1)$, $\Pr(S_{t+1}|H_t, A_t)$, $\Pr(A_t|S_t)$, and $\Pr(H_t|S_t, A_t)$. This leads to the following definition:

Definition 1. *An SF model (SFM) is a tuple* $\lambda \equiv (\mathbf{D}^a, \mathbf{K}^a, \mathbf{\Pi}, \boldsymbol{\mu})$, *where* $\mathbf{D}^a_{ij} = \Pr(H_t = j|S_t = i, A_t = a)$, $\mathbf{K}^a_{ij} = \Pr(S_{t+1} = j|H_t = i, A_t = a)$, $\mathbf{\Pi}_{ia} = \Pr(A_t = a|S_t = i)$, *and* $\boldsymbol{\mu}_i = \Pr(S_1 = i)$.

An SFM λ provides a useful construct for computing the factorization of an MDP from data. Let $Z_{1:\tau}$ be the "observable part" of $W_{1:\tau}$, that is, $Z_{1:\tau} \equiv \{S_1, A_1, ..., A_{\tau-1}, S_\tau\}$. The idea is to assume that $Z_{1:\tau}$ comes from an SFM λ and then tune the parameters of the model in order to maximize the likelihood of the data:

$$\mathcal{L}(\lambda|z_{1:\tau}) = \Pr(z_{1:\tau}|\lambda) = \boldsymbol{\mu}_{s_1}\prod_{t=1}^{\tau-1}\mathbf{\Pi}_{s_t a_t}(\mathbf{D}^{a_t}\mathbf{K}^{a_t})_{s_t s_{t+1}}. \tag{1}$$

We have shown that one can find matrices $\mathbf{D}^a$ and $\mathbf{K}^a$ corresponding to a local maximum of (1) by successively applying the following update rules:

$$\begin{gathered}\mathbf{D}^{a'}_{ij} = \textstyle\sum_{t=1}^{\tau-1}\Pr(H_t = j|Z_{1:\tau} = z_{1:\tau}, \lambda)\mathbf{1}\{s_t = i, a_t = a\},\\ \mathbf{K}^{a'}_{ij} = \textstyle\sum_{t=1}^{\tau-1}\Pr(H_t = i|Z_{1:\tau} = z_{1:\tau}, \lambda)\mathbf{1}\{s_{t+1} = j, a_t = a\},\\ \mathbf{D}^a_{ij} = \mathbf{D}^{a'}_{ij}/\textstyle\sum_l \mathbf{D}^{a'}_{il}, \text{ and } \mathbf{K}^a_{ij} = \mathbf{K}^{a'}_{ij}/\textstyle\sum_l \mathbf{K}^{a'}_{il},\end{gathered} \tag{2}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function. The question that naturally arises is of course how to compute $\Pr(H_t|Z_{1:\tau},\lambda)$. For that, we proposed an iterative method, inspired on the forward-backward procedure to compute probabilities in a Hidden Markov Model (Baum 1972), which works by propagating auxiliary variables $\alpha_i(t)$ and $\beta_i(t)$ through time. Specifically, we showed that, starting from $\alpha_i(1) = \boldsymbol{\mu}_{s_1}\boldsymbol{\Pi}_{s_1 a_1}\mathbf{D}^{a_1}_{s_{1i}}$ and $\beta_i(\tau-1) = \mathbf{K}^{a_{\tau-1}}_{is_\tau}$, by applying the recursive equations

$$\begin{aligned}\alpha_j(t+1) &= \textstyle\sum_i \alpha_i(t)\mathbf{K}^{a_t}_{is_{t+1}}\boldsymbol{\Pi}_{s_{t+1}a_{t+1}}\mathbf{D}^{a_{t+1}}_{s_{t+1}j}\\ \beta_i(t-1) &= \textstyle\sum_j \beta_j(t)\mathbf{D}^{a_t}_{s_t j}\boldsymbol{\Pi}_{s_t a_t}\mathbf{K}^{a_{t-1}}_{i,s_t}\end{aligned} \tag{3}$$

one can compute $\Pr(H_t = i|Z_{1:\tau},\lambda)$ for any t and any i as the product of a scaled version of $\alpha_i(t)$ and $\beta_i(t)$. The resulting algorithm is *expectation-maximization SF* (EMSF).

3 A Closer Look at EMSF

In this section we derive simplified rules for EMSF, analyze the divergence function it implicitly minimizes, and draw a connection with PLSA and NMF.

Simplified Update Rules

By exploiting Assumptions (i), (ii), and (iii) it is possible to break the dependence of $\Pr(H_t|Z_{1:\tau},\lambda)$ on events that happened before time t or after time $t+1$. This eliminates the need for the recursive computations in (3), leading to particularly simple instances of update equations (2). The following result states that the distribution of H_t only depends on S_t, A_t, and S_{t+1}:

Proposition 1. *Under Assumptions (i), (ii), and (iii)* $\Pr(H_t|Z_{1:\tau}) = \Pr(H_t|S_{t+1}, A_t, S_t)$.

Proof. (Sketch) We start from

$$\Pr(H_t|S_t, A_t, S_{t+1}) = \frac{\Pr(S_{t+1}|H_t, A_t)\Pr(H_t|S_t, A_t)}{\Pr(S_{t+1}|S_t, A_t)} \tag{4}$$

and show that if we expand $\Pr(H_t, Z_{1:\tau})$ and $\Pr(Z_{1:\tau})$ we find that $\Pr(H_t|S_{1:\tau}) = \Pr(H_t, S_{1:\tau})/\Pr(S_{1:\tau})$ reduces to (4). The detailed proof is in the supplementary material (Barreto et al. 2016). □

We can now use Proposition 1 in update equations (2). We start by rewriting (4) using the parameters of an SFM:

$$\Pr(H_t = j|Z_{1:\tau},\lambda) = \frac{\mathbf{D}^{a_t}_{s_t j}\mathbf{K}^{a_t}_{js_{t+1}}}{\sum_l \mathbf{D}^{a_t}_{s_t l}\mathbf{K}^{a_t}_{ls_{t+1}}} = \frac{\mathbf{D}^{a_t}_{s_t j}\mathbf{K}^{a_t}_{js_{t+1}}}{(\mathbf{D}^{a_t})_{s_t}(\mathbf{K}^{a_t})^{s_{t+1}}}. \tag{5}$$

We can then plug (5) in (2) to obtain our improved update rules. We define u matrices $\mathbf{C}^a$ where $\mathbf{C}^a_{ij}$ is the number of transitions in which $s_t = i$, $a_t = a$, and $s_{t+1} = j$. Then,

$$\begin{aligned}\mathbf{D}^{a'}_{ij} &= \textstyle\sum_{t=1}^{\tau-1}\mathbf{1}\{s_t = i, a_t = a\}\frac{\mathbf{D}^a_{ij}\mathbf{K}^a_{js_{t+1}}}{(\mathbf{D}^a)_i(\mathbf{K}^a)^{s_{t+1}}}\\ &= \textstyle\sum_{l=1}^{n}\sum_{t=1}^{\tau-1}\mathbf{1}\{s_t = i, a_t = a, s_{t+1} = l\}\frac{\mathbf{D}^a_{ij}\mathbf{K}^a_{jl}}{(\mathbf{D}^a)_i(\mathbf{K}^a)^l}\\ &= \mathbf{D}^a_{ij}\sum_{l=1}^{n}\frac{\mathbf{C}^a_{il}\mathbf{K}^a_{jl}}{(\mathbf{D}^a\mathbf{K}^a)_{il}}.\end{aligned} \tag{6}$$

If we proceed in the same way starting from the update equation for $\mathbf{K}^{a'}_{ij}$, we arrive at the following update rule:

$$\mathbf{K}^{a'}_{ij} = \mathbf{K}^a_{ij}\sum_{l=1}^{n}\frac{\mathbf{C}^a_{lj}\mathbf{D}^a_{li}}{(\mathbf{D}^a\mathbf{K}^a)_{lj}}. \tag{7}$$

Equations (6) and (7) improve (3) in at least three ways. First, they are easier and faster to compute (we discuss the precise computational costs in Section 4). Second, they allow one to learn an SFM without knowledge of the policy $\boldsymbol{\Pi}$ and initial distribution $\boldsymbol{\mu}$ used to collect the data. Third, they make it clear that the updates of $\mathbf{D}^a$ and $\mathbf{K}^a$ only depend on the transitions in which the action a was selected, and thus the model update can be broken in u independent subproblems. On top of that, (6) and (7) highlight a nice feature of EMSF: since any elements $\mathbf{D}^a_{ij}$ and $\mathbf{K}^a_{ij}$ that are initially zero will remain zero throughout its iterations, one can force the resulting model to be sparse, either for computational reasons or as a way of introducing prior knowledge in the approximation. For example, if we know that the execution of action a in state i cannot possibly lead to some states, we can zero the corresponding entries in some rows of $\mathbf{K}^a$ and define $\mathbf{D}^a_i$ accordingly: $d^a_{ij} \neq 0$ only if $\mathbf{K}^a_j$ is one of the zeroed rows.

Divergence Function Being Minimized by EMSF

Update rules (6) and (7) make it possible to compute an SF directly from data. Alternatively, one could use the sample transitions to compute an estimate of $\mathbf{P}^a$, let it be denoted by $\hat{\mathbf{P}}^a$, and then resort to a NMF algorithm to compute $\mathbf{D}^a$ and $\mathbf{K}^a$. In fact, Cohen and Rothblum (1991) have shown that one can always derive an SF from a NMF of a stochastic matrix; thus, in principle, *any* NMF algorithm can be used.

A natural question is thus how to choose between EMSF and one of the many NMF algorithms available in the literature (Wang and Zhang 2013). NMF algorithms minimize different *divergence functions*, and some are more appropriate than others depending on the application. In this section we show that maximization of (1) corresponds to the minimization of a specific divergence function that is well suited for learning an SF from data, which indicates that EMSF may indeed be a good choice.

Equations (6) and (7) resemble Lee and Seung's (1999) multiplicative update rules that helped to popularize NMF, in particular the ones used to minimize the Kullback-Leibler (KL) divergence:

$$\begin{aligned}\mathrm{KL}(\hat{\mathbf{P}}^a||\mathbf{D}^a\mathbf{K}^a) &= \sum_{i=1}^{n}\mathrm{KL}(\hat{\mathbf{P}}^a_i, (\mathbf{D}^a\mathbf{K}^a)_i)\\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}\left(\hat{\mathbf{P}}^a_{ij}\log\frac{\hat{\mathbf{P}}^a_{ij}}{(\mathbf{D}^a\mathbf{K}^a)_{ij}}\right).\end{aligned} \tag{8}$$

We now show that EMSF minimizes a modified version of (8) which is particularly adequate for learning an SF. As an EM method, EMSF aims at maximizing (1), which is

equivalent to maximizing

$$\log \mathcal{L}(\lambda|z_{1:\tau}) = \log \boldsymbol{\mu}_{s_1} + \sum_{t=1}^{\tau-1} \log \boldsymbol{\Pi}_{s_t,a_t} + \\ + \sum_{t=1}^{\tau-1} \log(\mathbf{D}^{a_t}\mathbf{K}^{a_t})_{s_t s_{t+1}}. \quad (9)$$

Since $\boldsymbol{\mu}$ and $\boldsymbol{\Pi}$ are not estimated by EMSF, we can exclude these fixed terms from (9) and write

$$\begin{aligned}\log \mathcal{L}(\lambda|z_{1:\tau}) &\propto \sum_{a=1}^{u}\sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{C}^a_{ij} \log(\mathbf{D}^a\mathbf{K}^a)_{ij} \\ &= \sum_{a=1}^{u}\sum_{i=1}^{n}\sum_{l} \mathbf{C}^a_{il} \sum_{j=1}^{n} \frac{\mathbf{C}^a_{ij}}{\sum_l \mathbf{C}^a_{il}} \log(\mathbf{D}^a\mathbf{K}^a)_{ij} \\ &= \sum_{a=1}^{u}\sum_{i=1}^{n}\sum_{l=1}^{n} \mathbf{C}^a_{il} \sum_{j=1}^{n} \hat{\mathbf{P}}^a_{ij} \log(\mathbf{D}^a\mathbf{K}^a)_{ij}, \quad (10)\end{aligned}$$

where $\hat{\mathbf{P}}^a$ is an estimate of $\mathbf{P}^a$ computed through counting, as in (8). The term $\sum_{j=1}^{n} \hat{\mathbf{P}}^a_{ij} \log(\mathbf{D}^a\mathbf{K}^a)_{ij}$ is the negative of the cross entropy between $\hat{\mathbf{P}}^a_i$ and $(\mathbf{D}^a\mathbf{K}^a)_i$, denoted by $\mathrm{H}(\hat{\mathbf{P}}^a_i, (\mathbf{D}^a\mathbf{K}^a)_i)$. It is well known that $\mathrm{H}(\hat{\mathbf{P}}^a_i, (\mathbf{D}^a\mathbf{K}^a)_i) = \mathrm{H}(\hat{\mathbf{P}}^a_i) + \mathrm{KL}(\hat{\mathbf{P}}^a_i||(\mathbf{D}^a\mathbf{K}^a)_i)$, where $\mathrm{H}(\hat{\mathbf{P}}^a_i)$ is the entropy of $\hat{\mathbf{P}}^a_i$ (Bishop 2006). Since $\mathrm{H}(\hat{\mathbf{P}}^a_i)$ is fixed in $\mathrm{H}(\hat{\mathbf{P}}^a_i, (\mathbf{D}^a\mathbf{K}^a)_i)$, this term plays no rule in the maximization of (10). Therefore, if we divide (10) by $\tau - 1$, we see that maximizing (9) corresponds to minimizing the following divergence function:

$$\mathrm{WKL} = \sum_{a=1}^{u}\sum_{i=1}^{n} \frac{\sum_{l=1}^{n} \mathbf{C}^a_{il}}{\tau - 1} \mathrm{KL}(\hat{\mathbf{P}}^a_i||(\mathbf{D}^a\mathbf{K}^a)_i). \quad (11)$$

We can look at $\sum_{l=1}^{n} \mathbf{C}^a_{il}/(\tau-1)$ as an estimate of $\Pr(S_t = i, A_t = a)$ under the stochastic process induced by $\boldsymbol{\mu}$ and $\boldsymbol{\Pi}$. Thus, minimization of (11) can be seen as an empirical minimization of

$$\mathrm{KL}_{(\boldsymbol{\mu},\boldsymbol{\Pi})} = \mathrm{E}_{(i,a)\sim(\boldsymbol{\mu},\boldsymbol{\Pi})} \left[\mathrm{KL}(\hat{\mathbf{P}}^a_i||(\mathbf{D}^a\mathbf{K}^a)_i)\right], \quad (12)$$

where $\mathrm{E}_{(i,a)\sim\boldsymbol{\mu},\boldsymbol{\Pi}}[\cdot]$ denotes expected value when $i \in \mathcal{S}$ and $a \in \mathcal{A}$ come from the distribution induced by $\boldsymbol{\mu}$ and $\boldsymbol{\Pi}$.

Therefore, the divergence function minimized by EMSF is a weighted version of (8). The difference may seem subtle at first, but the fact that (12) takes into account the actual distribution from which the data is coming may be a great advantage in some cases. To see why this is so, consider the scenario where the distribution induced by $\boldsymbol{\mu}$ and $\boldsymbol{\Pi}$ makes the occurrence of the event $S_t = i, A_t = a$ very unlikely (this is often the case in reinforcement learning). If we are minimizing (8) or (11) using a finite dataset, one would expect that the estimate $\hat{\mathbf{P}}^a_i$ is not a very good approximation of the i^{th} row of $\mathbf{P}^a$. Since (8) weighs the approximation of all the rows of $\mathbf{P}^a$ equally, by minimizing this function one will "waste resources" in the attempt to approximate a bad estimate of the true $\mathbf{P}^a_i$. In contrast, EMSF will essentially ignore $\mathbf{P}^a_i$, using the free parameters of $\mathbf{D}^a$ and $\mathbf{K}^a$ to approximate the rows of $\mathbf{P}^a$ that actually occur under $\boldsymbol{\mu}$ and $\boldsymbol{\Pi}$ (the precise way this happens will become clear shortly).

Relation with NMF and PLSA

Interestingly, it has been shown in the literature that (11) is the divergence function minimized by the asymmetric formulation of PLSA (Shashanka, Raj, and Smaragdis 2008). In fact, if we appropriately combine the three update equations usually used in PLSA we arrive precisely at (6) and (7); thus, from a strictly algorithmic point of view this model is a particular case of EMSF when $u = 1$.

PLSA is a statistical technique for modeling the co-occurrence of data by associating latent variables with observations (Hofmann 1999). Although it has been originally motivated by the analysis of text, it can be applied to arbitrary count data. In contrast with SF, though, in PLSA the variables being analyzed usually have different semantics—such as words and documents, for example—and the sampling process is assumed to be independent draws from an underlying distribution.

Besides being interesting on its own, the connection between EMSF and PLSA is beneficial because it immediately makes available many results regarding the latter. It clarifies, for example, the precise relationship between EMSF and NMF: it can be shown that rules (6) and (7) are numerically equivalent to NMF using (8) and applied to $1/(\tau-1)\mathbf{C}^a$ instead of $\hat{\mathbf{P}}^a$ (Gaussier and Goutte 2005).

More generally, the connection between EMSF and PLSA helps to contextualize the former within the framework of discrete component analysis, which in turn highlights a link with other models that are also potentially useful in the context of SF (Buntine and Jakulin 2006). From a more practical point of view, the equivalence between the two algorithms suggests several possible extensions for EMSF, such as Bayesian methods to determine an appropriate value for m (Mclachlan and Peel 2000), techniques to avoid overfitting (Hofmann 1999) and to induce sparsity (Shashanka, Raj, and Smaragdis 2008), and hierarchical variants of the algorithm (Vinokourov and Girolami 2002; Gaussier et al. 2002).

However, in this paper we are interested in one particular extension of EMSF, namely, an algorithm able to compute an SF approximation of an MDP online. Given the connection between EMSF and PLSA/NMF, it is only natural to consult the literature on the latter in search of online versions of the basic algorithms.

There is indeed a recent trend to develop PLSA and NMF algorithms that can operate online (Lefevre, Bach, and Févotte 2011; Duan, Mysore, and Smaragdis 2012; Bassiou and Kotropoulos 2014). Unfortunately, these algorithms are not readily applicable to the scenario considered by EMSF. In PLSA and NMF, each row of $\mathbf{P}$ represents an object, such as a document, image, or genome. The online algorithms were thus developed to handle a very long (or even infinite) stream of objects, which corresponds to having access to the rows of $\mathbf{P}$ incrementally. In contrast, when we are trying to approximate an MDP online *any* element of $\hat{\mathbf{P}}^a$ can change with the arrival of new data. This reflects the difference on the generative models underlying the two frameworks: while in PLSA "new data" is an entire object, in EMSF it is a single co-occurrence of two events—in the

case of an MDP, a sample transition.

In the next section we build on the simplified EMSF update rules to develop a new version of the algorithm which is able to process data incrementally, in the "atomic" sense explained above. Translating back to the terminology usually adopted in PLSA, our algorithm is able to process a stream of *words*, each one associated with a given document. To the best of our knowledge this is the first method that allows such an asynchronous analysis of objects, so the algorithm we are about to present can be considered as a contribution to the areas of PLSA and NMF as well.

4 Incremental Algorithm

We start the derivation of the new algorithm by rewriting (6) and (7) in terms of the rows of $\mathbf{D}^a$ and columns of $\mathbf{K}^a$. Specifically, given auxiliary matrices $\hat{\mathbf{D}}^a$ and $\hat{\mathbf{K}}^a$, for each $\mathbf{C}^a_{ij} \neq 0$ we simply make

$$\mathbf{w}^a_{ij} \leftarrow \frac{(\mathbf{D}^a)_i \otimes (\mathbf{K}^a)^j}{(\mathbf{D}^a)_i (\mathbf{K}^a)^j},$$
$$\hat{\mathbf{D}}^a_i \leftarrow \hat{\mathbf{D}}^a_i + \mathbf{C}^a_{ij}\mathbf{w}^a_{ij} \text{ and } (\hat{\mathbf{K}}^a)^j \leftarrow (\hat{\mathbf{K}}^a)^j + \mathbf{C}^a_{ij}\mathbf{w}^a_{ij}, \tag{13}$$

and after every nonzero element of $\mathbf{C}^a$ has been processed matrices $\hat{\mathbf{D}}^a$ and $\hat{\mathbf{K}}^a$ are normalized and substituted for $\mathbf{D}^a$ and $\mathbf{K}^a$ (here '$\otimes$' denotes element-wise multiplication).

Update rules (13) make it explicit the advantages of the new version of EMSF in terms of computational resources required. Let η^a be the number of nonzero elements in matrix $\mathbf{C}^a$ and let $\eta = \max_a \eta^a$. Each iteration of EMSF has computational complexity $O(\eta u m)$, while the algorithm's memory complexity is $O(u(\eta + nm))$. Note that, unlike in the original version of EMSF, in which both the amount of memory used and the number of operations performed are $O(\tau m)$, in the new version of the algorithm these quantities do not depend on the number of transitions τ (except if we consider the construction of $\mathbf{C}^a$, which is $O(\tau)$).

As discussed in Section 3, $\mathbf{w}^a_{ij}$ is an m-dimensional vector that represents $\Pr(H_t | S_t = i, A_t = a, S_{t+1} = j)$. One can look at this vector as a distribution reflecting the "intersection" between $\mathbf{D}^a_i$ and $(\mathbf{K}^a)^j$. Thus, each row of $\mathbf{D}^a$ and each column of $\mathbf{K}^a$ will be updated as a weighted sum of these stochastic vectors in which the weight of $\mathbf{w}^a_{ij}$ is $\mathbf{C}^a_{ij}$, the number of times $S_t = i$, $A_t = a$, and $S_{t+1} = j$. It is in this sense that EMSF allocates more resources to transitions that actually occur under $\boldsymbol{\mu}$ and $\boldsymbol{\Pi}$; in particular, if $\mathbf{C}^a_{ij}$ is close to zero, the corresponding $\mathbf{w}^a_{ij}$ will have a small influence on the update of $\mathbf{D}^a_i$ and $(\mathbf{K}^a)^j$.

Besides shedding light on its mechanics, this view of EMSF provides a very flexible framework that can be specialized to different contexts. In particular, if we decouple the accumulation of changes to $\mathbf{D}^a$ and $\mathbf{K}^a$ from their committing, we can define an incremental version of EMSF.

Algorithm 1 shows the incremental version of EMSF. At each iteration the algorithm gets a new sample transition, either from a finite dataset or from direct interaction with the MDP. The transition is then added to the appropriate matrix of countings $\mathbf{C}^a$ and discarded. If the sum of nonzero elements in the matrices $\mathbf{C}^a$ reaches a certain limit $\eta_{\max}$, defined according to the amount of memory available, the statistics stored in $\mathbf{C}^a$ are used to compute the updates to $\mathbf{D}^a$ and $\mathbf{K}^a$, which are accumulated in the auxiliary matrices $\hat{\mathbf{D}}^a$ and $\hat{\mathbf{K}}^a$, as in (13). At every t_c iterations the modifications in $\hat{\mathbf{D}}^a$ and $\hat{\mathbf{K}}^a$ are committed to $\mathbf{D}^a$ and $\mathbf{K}^a$.

Algorithm 1 Incremental EMSF

Input: $m \in \mathbb{N}$ ▷ SF's order; $t_c \in \mathbb{N}$ ▷ Interval to commit updates; $\alpha \in (0, 1]$ ▷ Learning rate; $\eta_{\max} \in \mathbb{N}$ ▷ Max. number of nonzero elements

Output: $\mathbf{D}^a\mathbf{K}^a \approx \mathbf{P}^a$, for $a \in \mathcal{A}$

for each $a \in \mathcal{A}$ **do**
- $\mathbf{D}^a \leftarrow$ random stochastic matrix $\in \mathbb{R}^{n \times m}$
- $\mathbf{K}^a \leftarrow$ random stochastic matrix $\in \mathbb{R}^{m \times n}$
- $\mathbf{C}^a \leftarrow \mathbf{0} \in \mathbb{R}^{n \times n}$ ▷ Matrix with countings
- $\hat{\mathbf{D}}^a \leftarrow \mathbf{0} \in \mathbb{R}^{n \times m}$ ▷ Auxiliary matrix
- $\hat{\mathbf{K}}^a \leftarrow \mathbf{0} \in \mathbb{R}^{m \times n}$ ▷ Auxiliary matrix
- $\hat{\mathbf{x}}^a \leftarrow \mathbf{0} \in \mathbb{R}^n$ ▷ Sums of $\hat{\mathbf{D}}^a$'s rows
- $\hat{\mathbf{y}}^a \leftarrow \mathbf{0} \in \mathbb{R}^m$ ▷ Sums of $\hat{\mathbf{K}}^a$'s rows

for $t = 1, 2, ...$ **do**
- get next s, a, s' ▷ From dataset or online
- $\mathbf{C}^a_{ss'} \leftarrow \mathbf{C}^a_{ss'} + 1$ ▷ Count transition
- $\mathrm{ct} \leftarrow (t \bmod t_c = 0)$ ▷ ct stands for "commit time"
- **if** $\sum_a \eta^a = \eta_{\max}$ **or** ct **then** ▷ Accumulate changes
  - **for** each $a \in \mathcal{A}$ **do**
    - **for** each $\mathbf{C}^a_{ij} \neq 0$ **do**
      - $g \leftarrow \mathbf{D}^a_i (\mathbf{K}^a)^j$
      - $\mathbf{w} \leftarrow (\mathbf{C}^a_{ij}/g) \times (\mathbf{D}^a_i \otimes (\mathbf{K}^a)^j)$
      - $(\hat{\mathbf{D}}^a)_i \leftarrow (\hat{\mathbf{D}}^a)_i + \mathbf{w}$
      - $\hat{\mathbf{x}}^a_i \leftarrow \hat{\mathbf{x}}^a_i + \sum_l \mathbf{w}_l$
      - $(\hat{\mathbf{K}}^a)^j \leftarrow (\hat{\mathbf{K}}^a)^j + \mathbf{w}$
      - $\hat{\mathbf{y}}^a \leftarrow \hat{\mathbf{y}}^a + \mathbf{w}$
    - $\mathbf{C}^a \leftarrow \mathbf{0} \in \mathbb{R}^{n \times n}$ ▷ Free memory
- **if** ct **then** ▷ Commit changes
  - **for** each $a \in \mathcal{A}$ **do**
    - **for** $i \in 1, 2, ..., m$ **do** $\hat{\mathbf{K}}^a_i \leftarrow \hat{\mathbf{K}}^a_i / \hat{\mathbf{y}}^a_i$
    - $\mathbf{K}^a \leftarrow (1 - \alpha)\mathbf{K}^a + \alpha\hat{\mathbf{K}}^a$
    - **for** $i \in 1, 2, ..., n$ **do**
      - **if** $\hat{\mathbf{x}}^a_i \neq 0$ **then** ▷ If row i changed
        - $\mathbf{D}^a_i \leftarrow (1 - \alpha)\mathbf{D}^a_i + \alpha\hat{\mathbf{D}}^a_i / \hat{\mathbf{x}}^a_i$
    - $\hat{\mathbf{D}}^a \leftarrow \mathbf{0} \in \mathbb{R}^{n \times m}$; $\hat{\mathbf{K}}^a \leftarrow \mathbf{0} \in \mathbb{R}^{m \times n}$
    - $\hat{\mathbf{x}}^a \leftarrow \mathbf{0} \in \mathbb{R}^m$; $\hat{\mathbf{y}}^a \leftarrow \mathbf{0} \in \mathbb{R}^n$
  - Update policy used to collect data ▷ Optional

The scheme above can give rise to different instantiations of EMSF. For example, the case in which $t_c = \tau - 1$, $\alpha = 1$, and $\eta_{\max} \geq un^2$ corresponds exactly to the batch version of EMSF, which we know converges to a stationary point of (9) and (11) (Bishop 2006). Here though we are mostly interested in the scenario where we have to process a very long or infinite sequence of transitions, so learning should start *before* all the data is processed. On top of that, we also consider the additional difficulty of handling matrices whose

size n is large compared to the amount of memory available.

We deal with the memory issue first. The need to keep $O(u(\eta + nm))$ elements makes it infeasible to store $\mathbf{C}^a$ when this matrix is dense and n is large. We can solve this problem by only partially filling the matrices $\mathbf{C}^a$ and then deleting them from memory after the corresponding updates have been accumulated in the auxiliary matrices $\hat{\mathbf{D}}^a$ and $\hat{\mathbf{K}}^a$. This can be accomplished with Algorithm 1 by setting the parameter $\eta_{\max}$, the maximum number of nonzero elements in the matrices $\mathbf{C}^a$, so that these matrices will fit in the available memory. Observe that this strategy breaks the memory complexity's dependency on η, which means that EMSF's memory usage can be reduced to $O(unm)$. Despite this fact, when $t_c = \tau - 1$ EMSF will converge to the exact same solution as its batch counterpart, though using a larger number of operations, as each application of (13) will be broken in several rounds. When $\eta_{\max} = 1$, in particular, EMSF will perform roughly the same number of operations performed by its original version. More generally, by using $\eta_{\max} < un^2$ one is simply trading space for computation time, since the parameter $\eta_{\max}$ does not affect the algorithm's result.

Since the issue with large n has been solved through the parameter $\eta_{\max}$, we now turn to the second challenge: how to make EMSF work online. This is where the parameter t_c comes into play. Specifically, by making $t_c < \tau - 1$, the matrices $\mathbf{D}^a$ and $\mathbf{K}^a$ can be updated at arbitrary time steps. It is not difficult to see that update rules (13) only depend on the relative magnitude of the elements of $\mathbf{C}^a$; thus, as long as t_c is large enough, these rules will have the expected effect. In particular, as $t_c \to \infty$ the solution found by EMSF should approach the solution that would be computed in the limiting case of an infinite sample—*i.e.*, when $\mathbf{C}^a_{ij}/(\tau - 1) = \Pr(S_t = i, A_t = a, S_{t+1} = j)$.

An interesting modification of EMSF that can potentially speed up convergence is to let the algorithm commit the changes accumulated in $\hat{\mathbf{D}}^a$ and $\hat{\mathbf{K}}^a$ at shorter time intervals t_c. Since in this case the auxiliary matrices may not be "fully formed" at committing time, such modification may lead to increased variance on EMSF's results. One alternative in this case is to use $\alpha < 1$ in Algorithm 1. Roughly, this modification corresponds to changing from batch-mode learning to a stochastic approximation regime, as commonly done in neural networks training (Bishop 2006). The convergence of the algorithm should occur under the usual conditions imposed on α for the convergence of stochastic approximation techniques (Bertsekas and Tsitsiklis 1996). In the case in which $\alpha < 1$ and $\eta_{\max} = t_c = 1$ Algorithm 1 reduces to the instance of EM proposed by Nowlan (1991).

5 Experiments

In this section we use computational experiments to illustrate some of the properties of EMSF. Since PLSA/NMF algorithms similar to EMSF have already been submitted to extensive empirical analysis (Wang and Zhang 2013), we focus on illustrating characteristics that are specific to EMSF.

We argued that one potential advantage of EMSF is that the divergence function it minimizes is particularly suitable for learning an SF from data because it promotes a more rational allocation of resources. We now test this hypothesis by comparing the batch version of EMSF with Lee and Seung's (1999) NMF algorithm that minimizes the conventional KL divergence (8) (we call the method KLM). In order to check our hypothesis, we carried out an experiment in which 10^5 transitions were sampled from $\mathbf{P}$ in two different regimes. In the first one the rows of $\mathbf{P}$ were sampled from a uniform distribution ρ (that is, we first sample a state i uniformly from $\mathcal{S}$ and then sample the next state from $\mathbf{P}_i$). This corresponds to weighing all rows $\mathbf{P}_i$ the same, as in (8). In the second experiment we concentrated 90% of the probability mass of ρ in 50% of the states (we did so by simply renormalizing ρ). This heterogeneous sampling scheme emulates situations in which some states are much more likely to be sampled than others, such as in reinforcement learning. The results of the experiments, shown in Figure 2, illustrate how the approximation computed by EMSF takes into account the distribution used to sample the data.

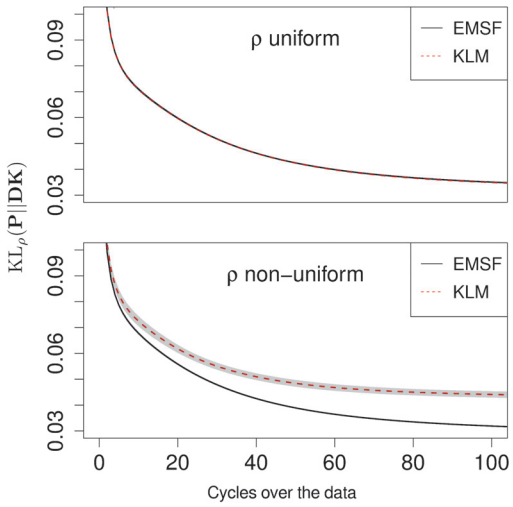

Figure 2: Approximation error for transition matrices generated as $\mathbf{P} = \mathbf{D}'\mathbf{K}'$, with the rows of $\mathbf{D}' \in \mathbb{R}^{100\times 20}$ and $\mathbf{K}' \in \mathbb{R}^{20\times 100}$ sampled from a Dirichlet distribution with concentration parameter set to 0.5. Both algorithms used $m = 10$; EMSF was run with $t_c = \tau - 1$ and $\alpha = 1$. Shadowed regions represent one standard error over 50 runs.

We now analyze the online performance of EMSF by executing Algorithm 1 with different values of t_c and α (recall that $\eta_{\max}$ is only a way to balance memory usage and execution time, not affecting the solution computed by the algorithm). In this experiment learning took place with transitions sampled through direct interaction with $\mathbf{P}$, which corresponds to collecting data under its stationary distribution. The results of the experiment, shown in Figure 3, corroborate the hypothesis that values of $\alpha < 1$ can indeed speed up convergence. They also illustrate an interesting interplay between this parameter and t_c: note that, when using larger values for t_c, we can afford to use larger values for α as well. This is expected, since we know that for sufficiently large t_c EMSF will converge to a stationary point of (11)

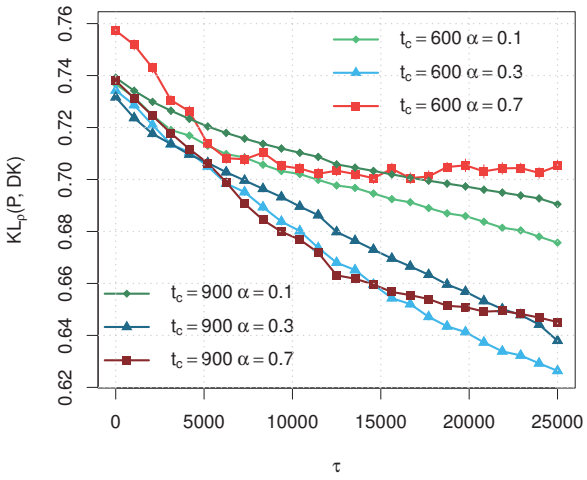

Figure 3: $\mathrm{KL}_\rho(\mathbf{P}||\mathbf{DK})$, with $n = 100$, $m = 30$, and ρ the stationary distribution of $\mathbf{P}$. The rows of $\mathbf{P}$ were generated from a Dirichlet distribution with concentration parameter 0.5. Error bars representing one standard error over 50 runs are shown, but are almost imperceptible at the plot's scale.

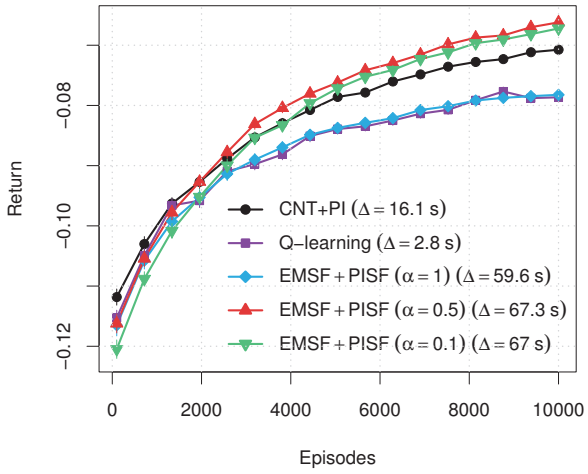

Figure 4: Results on blackjack. The values are the empirical return obtained on 10^6 games; Δ is the execution time. EMSF was run with $t_c = 100$ episodes and $\eta_{\max} = \infty$. Error bars representing one standard error over 100 runs are shown, but are almost imperceptible at the plot's scale.

using $\alpha = 1$.

Finally, we investigate the performance of EMSF in the context of reinforcement learning. In order to do so, we replicated our previous experiments with the game of blackjack, but now considering the online regime (Barreto et al. 2015). The experiment was carried out as follows. Starting with a random policy π, each algorithm collected a batch of 100 episodes. This data was used to improve π, which was then tested on 10^6 hands of blackjack. Next, an 0.15-greedy version of π was used to collect a second batch of 100 episodes, restarting the loop.

In order to evaluate EMSF we combined it with an algorithm called policy iteration based on stochastic factorization (PISF). Given $\mathbf{D}^a\mathbf{K} \approx \mathbf{P}^a$, PISF computes an approximation of the MDP's value function by applying the SF trick in each policy evaluation step (see the article by Barreto, Pineau, and Precup, 2014 for details). EMSF+PISF was compared with two alternatives. The first method keeps u approximations $\hat{\mathbf{P}}^a \in \mathbb{R}^{n\times n}$, computed through maximum-likelihood estimation —that is, counting transitions—, and uses policy iteration to determine π. We call this method "CNT+PI." The second algorithm is Q-learning using a learning rate of 0.1 (this rate resulted in the best performance among the values $\{0.01, 0.1, 0.3\}$).

The results on the game of blackjack are shown in Figure 4. It is clear from the figure that the model-based algorithms perform considerably better than Q-learning on this task. When we compare the model-based methods only, we see that EMSF+PISF can outperform CNT+PI using a model that is only 5% the size of the MDP. This suggests that the approximations $\mathbf{D}^a\mathbf{K} \approx \mathbf{P}^a$ computed by EMSF represent a better compromise between estimation and approximation errors (or "bias" and "variance"), as discussed in the original paper presenting the algorithm (Barreto et al. 2015).

Regarding computational cost, the results shown in Figure 4 suggest that EMSF+PISF is considerably slower than CNT+PI. Note though that there is a trade-off here: if on one hand EMSF spends more time than CNT to build the model, on the other hand PISF is significantly faster than PI. Specifically, in the worst case each iteration of EMSF is $O(n^2)$ and each iteration of PISF is $O(n)$, while CNT is $O(n)$ and PI is roughly $O(n^3)$ per iteration. The relative costs of CNT+PI and EMSF+PISF will thus depend on the number of iterations performed by each of their components, which will in turn have an impact on the quality of the approximations computed by the algorithms. How exactly these factors interact with each other is a matter yet to be investigated. We refer the reader to the supplement for additional experiments and further analysis (Barreto et al. 2016).

6 Conclusion

The contributions of this paper are twofold. First, it improves considerably our understanding of EMSF. By analyzing the algorithm, we were able to derive simple multiplicative update rules that supersede the original ones, and also uncover the divergence function implicitly minimized, which is particularly adequate for learning an SF from data sampled from an arbitrary distribution. We also established an interesting connection with PLSA and NMF that contextualizes EMSF and opens up possibilities of extensions. Building on this improved understanding of the algorithm we presented the paper's second contribution: an incremental version of EMSF that makes it possible to learn an SF concomitantly with the collection of data. This can speed up the convergence of the algorithm considerably, and makes it particularly suitable for online reinforcement learning.

Acknowledgments The authors thank Eduardo Krempser for helping with the experiments. Funding for this research was provided by *Conselho Nacional de Desenvolvimento*

Científico e Tecnológico (CNPq), grant 461739/2014-3, and by the NSERC Discovery grant program.

References

Barreto, A. M. S., and Fragoso, M. D. 2011. Computing the stationary distribution of a finite Markov chain through stochastic factorization. *SIAM Journal on Matrix Analysis and Applications* 32:1513–1523.

Barreto, A. M. S.; Beirigo, R. L.; Pineau, J.; and Precup, D. 2015. An expectation-maximization algorithm to compute a stochastic factorization from data. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 3329–3336.

Barreto, A. M. S.; Beirigo, R.; Pineau, J.; and Precup, D. 2016. Incremental stochastic factorization for online reinforcement learning: Supplementary material. Available online.

Barreto, A. M. S.; Pineau, J.; and Precup, D. 2014. Policy iteration based on stochastic factorization. *Journal of Artificial Intelligence Research* 50:763–803.

Bassiou, N., and Kotropoulos, C. 2014. Online PLSA: Batch updating techniques including out-of-vocabulary words. *IEEE Transactions on Neural Networks and Learning Systems* 25(11):1953–1966.

Baum, L. E. 1972. An inequality and associated maximization technique in statistical estimation of probabilistic functions of Markov processes. *Inequalities* 3:1–8.

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Bertsekas, D. P. 2011. Approximate policy iteration: a survey and some new methods. *Journal of Control Theory and Applications* 9(3):310–335.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. Springer-Verlag.

Buntine, W., and Jakulin, A. 2006. Discrete component analysis. In *Subspace, Latent Structure and Feature Selection*, volume 3940 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 1–33.

Cohen, J. E., and Rothblum, U. G. 1991. Nonnegative ranks, decompositions and factorizations of nonnegative matrices. *Linear Algebra and its Applications* 190:149–168.

Duan, Z.; Mysore, G. J.; and Smaragdis, P. 2012. Online PLCA for real-time semi-supervised source separation. In *Latent Variable Analysis and Signal Separation*, volume 7191 of *Lecture Notes in Computer Science*, 34–41. Springer Verlag.

Gaussier, E., and Goutte, C. 2005. Relation between PLSA and NMF and implications. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, 601–602.

Gaussier, . E.; Goutte, C.; Popat, K.; and Chen, F. 2002. A hierarchical model for clustering and categorising documents. In *Proceedings of the 24th BCS-IRSG European Colloquium on IR Research*, 229–247.

Györfi, L.; Kohler, M.; Krzyżak, A.; and Walk, H. 2002. *A Distribution-Free Theory of Nonparametric Regression*. Springer Verlag, New York.

Hofmann, T. 1999. Probabilistic latent semantic indexing. In *Proceedings of International ACM SIGIR Conference on Research and Development in Information Retrieval*, 50–57.

Lee, D. D., and Seung, H. S. 1999. Learning the parts of objects by non-negative matrix factorization. *Nature* 401:788–791.

Lee, D. D., and Seung, H. S. 2001. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems (NIPS)*. 556–562.

Lefevre, A.; Bach, F.; and Févotte, C. 2011. Online algorithms for nonnegative matrix factorization with the Itakura-Saito divergence. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 313–316.

Mclachlan, G., and Peel, D. 2000. *Finite Mixture Models*. Wiley-Interscience, 1 edition.

Nowlan, S. J. 1991. *Soft Competitive Adaptation: Neural Network Learning Algorithms Based on Fitting Statistical Mixtures*. Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA, USA.

Puterman, M. L. 1994. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Shashanka, M.; Raj, B.; and Smaragdis, P. 2008. Sparse overcomplete latent variable decomposition of counts data. In *Advances in Neural Information Processing Systems (NIPS)*. 1313–1320.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Vavasis, S. A. 2009. On the complexity of nonnegative matrix factorization. *SIAM Journal on Optimization* 20:1364–1377.

Vinokourov, A., and Girolami, M. 2002. A probabilistic framework for the hierarchic organisation and classification of document collections. *Journal of Intelligent Information Systems* 18(2-3):153–172.

Wang, Y.-X., and Zhang, Y.-J. 2013. Nonnegative matrix factorization: A comprehensive review. *IEEE Transactions on Knowledge and Data Engineering* 25(6).