Dictionary Learning for Massive Matrix Factorization

Arthur Mensch
Parietal team, Inria, CEA, Paris-Saclay University. Neurospin, Gif-sur-Yvette, France

ARTHUR.MENSCH@M4X.ORG

Julien Mairal
Thoth team, Inria, Grenoble, France

JULIEN.MAIRAL@INRIA.FR

Bertrand Thirion
Gaël Varoquaux
Parietal team, Inria, CEA, Paris-Saclay University. Neurospin, Gif-sur-Yvette, France

BETRAND.THIRION@INRIA.FR
GAE. VAROQUAUX@INRIA.FR

Abstract

Sparse matrix factorization is a popular tool to obtain interpretable data decompositions, which are also effective to perform data completion or denoising. Its applicability to large datasets has been addressed with online and randomized methods, that reduce the complexity in one of the matrix dimension, but not in both of them. In this paper, we tackle very large matrices in both dimensions. We propose a new factorization method that scales gracefully to terabyte-scale datasets. Those could not be processed by previous algorithms in a reasonable amount of time. We demonstrate the efficiency of our approach on massive functional Magnetic Resonance Imaging (fMRI) data, and on matrix completion problems for recommender systems, where we obtain significant speed-ups compared to state-of-the art coordinate descent methods.

As a widely-used model, the literature of matrix factorization is very rich and two main classes of formulations have emerged. The first one addresses an optimization problem involving a convex penalty, such as the trace or max norms (Srebro et al., 2004). These penalties promote low-rank structures, have strong theoretical guarantees (Candès & Recht, 2009), but they do not encourage sparse factors and lack scalability for very-large datasets. For these reasons, our paper is focused on a second type of approach, that relies on nonconvex optimization. Specifically, the motivation of our work originally came from the need to analyze huge-scale fMRI datasets, and the difficulty of current algorithms to process them.

To gain scalability, stochastic (or online) optimization methods have been developed; unlike classical alternate minimization procedures, they learn matrix decompositions by observing a single matrix column (or row) at each iteration. In other words, they stream data along one matrix dimension. Their cost per iteration is significantly reduced, leading to faster convergence in various practical contexts. More precisely, two approaches have been particularly successful: stochastic gradient descent (see Bottou, 2010) has been widely used in recommender systems (see Bell & Koren, 2007; Rendle & Schmidt-Thieme, 2008; Rendle, 2010; Blondel et al., 2015, and references therein), and stochastic majorization-minimization methods for dictionary learning with sparse and/or structured regularization (Mairal et al., 2010; Mairal, 2013). Yet, stochastic algorithms for dictionary learning are currently unable to deal efficiently with matrices that are large in both dimensions.

In a somehow orthogonal way, the growth of dataset size has proven to be manageable by randomized methods, that exploit random projections (Johnson & Lindenstrauss, 1984; Bingham & Mannila, 2001) to reduce data dimension.
The goal of matrix factorization is to decompose a matrix $X \in \mathbb{R}^{p \times n}$ — typically $n$ signals of dimension $p$ — as a product of two smaller matrices:

$$X \approx DA \quad \text{with} \quad D \in \mathbb{R}^{p \times k}, \ A \in \mathbb{R}^{k \times n},$$

with potential sparsity or structure requirements on $D$ and $A$. In statistical signal applications, this is often a dictionary learning problem, enforcing sparse coefficients $A$. In such a case, we call $D$ the “dictionary” and $A$ the sparse codes. We use this terminology throughout the paper.

Learning the dictionary is typically performed by minimizing a quadratic data-fitting term, with constraints and/or penalties over the code and the dictionary:

$$\min_{D \in \mathbb{C}} \sum_{i=1}^{n} \frac{1}{2} \|x_i - D\alpha_i\|^2_2 + \lambda \Omega(\alpha),$$

where $\mathbb{C}$ is a convex set of $\mathbb{R}^{p \times k}$, and $\Omega : \mathbb{R}^p \rightarrow \mathbb{R}$ is a penalty over the code, to enforce structure or sparsity. In large $n$ and large $p$ settings, typical in recommender systems, this problem is solved via block coordinate descent, which boils down to alternating least squares if regularization on $D$ and $\alpha$ are quadratic (Hastie et al., 2014).

**Constraints and penalties.** The constraint set $\mathbb{C}$ is traditionally a technical constraint ensuring that the coefficients $\alpha$ do not vanish, making the effect of the penalty $\Omega$ disappear. However, other constraints can also be used to enforce sparsity or structure on the dictionary (see Varouquaht et al., 2013). In our paper, $\mathbb{C}$ is the Cartesian product of a $\ell_1$ or $\ell_2$ norm ball:

$$\mathbb{C} = \{D \in \mathbb{R}^{p \times k} : \psi(d_j) \leq 1 \quad \forall j = 1, \ldots, k\},$$

where $D = [d_1, \ldots, d_k]$ and $\psi = \|\cdot\|_1$ or $\psi = \|\cdot\|_2$. The choice of $\psi$ and $\Omega$ typically offers some flexibility in the regularization effect that is desired for a specific problem; for instance, classical dictionary learning uses $\psi = \|\cdot\|_2$ and $\Omega = \|\cdot\|_1$, leading to sparse coefficients $\alpha$, whereas our experiments on fMRI uses $\psi = \|\cdot\|_1$ and $\Omega = \|\cdot\|_2$, leading to sparse dictionary elements $d_j$ that can be interpreted as brain activation maps.

1.2. Streaming Signals with Online Algorithms

In stochastic optimization, the number of signals $n$ is assumed to be large (or potentially infinite), and the dictionary $D$ can be written as a solution of

$$\min_{D \in \mathbb{C}} f(D) \quad \text{where} \quad f(D) = \mathbb{E}_x[l(x, D)]$$

$$l(x, D) = \min_{\alpha \in \mathbb{R}^k} \frac{1}{2} \|x - D\alpha\|^2_2 + \lambda \Omega(\alpha),$$

where the signals $x$ are assumed to be i.i.d. samples from an unknown probability distribution. Based on this formu-
loration, Mairal et al. (2010) have introduced an online dictionary learning approach that draws a single signal $x_t$ at iteration $t$ (or a minibatch), and computes its sparse code $\alpha_t$ using the current dictionary $D_{t-1}$ according to

$$\alpha_t \leftarrow \text{argmin}_{\alpha \in \mathbb{R}^k} \frac{1}{2} \|x_t - D_{t-1} \alpha\|_2^2 + \lambda \Omega(\alpha).$$

(5)

Then, the dictionary is updated by approximately minimizing the following surrogate function

$$g_t(D) = \frac{1}{t} \sum_{i=1}^t \frac{1}{2} \|x_i - D \alpha_i\|_2^2 + \lambda \Omega(\alpha_i),$$

(6)

which involves the sequence of past signals $x_1, \ldots, x_t$ and the sparse codes $\alpha_1, \ldots, \alpha_t$ that were computed in the past iterations of the algorithm. The function $g_t$ is called a "surrogate" in the sense that it only approximates the objective $f$. In fact, it is possible to show that it converges to a locally tight upper-bound of the objective, and that minimizing $g_t$ at each iteration asymptotically provides a stationary point of the original optimization problem. The underlying principle is that of majorization-minimization, used in a stochastic fashion (Mairal, 2013).

One key to obtain efficient dictionary updates is the observation that the surrogate $g_t$ can be summarized by a few sufficient statistics that are updated at every iteration. In other words, it is possible to describe $g_t$ without explicitly storing the past signals $x_i$ and codes $\alpha_i$ for $i \leq t$. Indeed, we may define two matrices $B_t \in \mathbb{R}^{p \times k}$ and $C_t \in \mathbb{R}^{k \times k}$

$$C_t = \frac{1}{t} \sum_{i=1}^t \alpha_i \alpha_i^\top, \quad B_t = \frac{1}{t} \sum_{i=1}^t x_i \alpha_i^\top,$$

(7)

and the surrogate function is then written:

$$g_t(D) = \frac{1}{2} \text{Tr}(D^\top D C_t - D^\top B_t) + \frac{\lambda}{t} \sum_{i=1}^t \Omega(\alpha_i).$$

(8)

The gradient of $g_t$ can be computed as

$$\nabla_D g_t(D) = D C_t - B_t.$$

(9)

Minimization of $g_t$ is performed using block coordinate descent on the columns of $D$. In practice, the following updates are successively performed by cycling over the dictionary elements $d_j$ for $j = 1, \ldots, k$

$$d_j \leftarrow \text{Proj}_{\psi(.) \leq 1} \left[ d_j - \frac{1}{C_t(j,j)} \nabla_d g_t(D) \right],$$

(10)

where Proj denotes the Euclidean projection over the constraint norm constraint $\psi$. It can be shown that this update corresponds to minimizing $g_t$ with respect to $d_j$ when fixing the other dictionary elements (see Mairal et al., 2010).

1.3. Handling Missing Values

Factorization of matrices with missing value have raised a significant interest in signal processing and machine learning, especially as a solution for recommender systems. In the context of dictionary learning, a similar effort has been made by Szabó et al. (2011) to adapt the framework to missing values. Formally, a mask $M$, represented as a binary diagonal matrix in $\{0, 1\}^{p \times p}$, is associated with every signal $x$, such that the algorithm can only observe the product $M x_t$ at iteration $t$ instead of a full signal $x_t$. In this setting, we naturally derive the following objective

$$\min_{D \in \mathbb{C}} f(D) \quad \text{where} \quad f(D) = \mathbb{E}_{x,M} [l(x, M, D)],$$

(11)

$$l(x, M, D) = \min_{\alpha \in \mathbb{R}^k} \frac{p}{2 \text{Tr} M} \|M (x - D \alpha)\|_2^2 + \lambda \Omega(\alpha),$$

where the pairs $(x, M)$ are drawn from the (unknown) data distribution. Adapting the online algorithm of Mairal et al. (2010) would consist of drawing a sequence of pairs $(x_t, M_t)$, and building the surrogate

$$g_t(D) = \frac{1}{t} \sum_{i=1}^t \frac{p}{2 s_i} \|M_t (x_i - D \alpha_i)\|_2^2 + \lambda \Omega(\alpha_i),$$

(12)

where $s_i = \text{Tr} M_t$ is the size of the mask and

$$\alpha_i \leftarrow \text{argmin}_{\alpha \in \mathbb{R}^k} \frac{p}{2 s_i} \|M_t (x_i - D_{t-1} \alpha)\|_2^2 + \lambda \Omega(\alpha).$$

(13)

Unfortunately, this surrogate cannot be summarized by a few sufficient statistics due to the masks $M_i$: some approximations are required. This is the approach chosen by Szabó et al. (2011). Nevertheless, the complexity of their update rules is linear in the full signal size $p$, which makes them unadapted to the large-$p$ regime that we consider.

2. Dictionary Learning for Massive Data

Using the formalism exposed above, we now consider the problem of factorizing a large matrix $X$ in $\mathbb{R}^{p \times n}$ into two factors $D$ in $\mathbb{R}^{p \times k}$ and $A$ in $\mathbb{R}^{k \times n}$ with the following setting: both $n$ and $p$ are large (greater than 100 000 up to several millions), whereas $k$ is reasonable (smaller than 1 000 and often near 100), which is not the standard dictionary-learning setting; some entries of $X$ may be missing. Our objective is to recover a good dictionary $D$ taking into account appropriate regularization.

To achieve our goal, we propose to use an objective akin to (11), where the masks are now random variables independent from the samples. In other words, we want to combine ideas of online dictionary learning with random subsampling, in a principled manner. This leads us to consider an infinite stream of samples $(M_t x_t)_{t \geq 0}$, where the signals $x_t$ are i.i.d. samples from the data distribution – that
Dictionary Learning for Massive Matrix Factorization

is, a column of $X$ selected at random – and $M_t$ “selects” a random subset of observed entries in $X$. This setting can accommodate missing entries, never selected by the mask, and only requires loading a subset of $x_t$ at each iteration.

The main justification for choosing this objective function is that in the large sample regime $p \gg k$ that we consider, computing the code $\alpha_t$ using only a random subset of the data $x_t$ according to (13) is a good approximation of the code that may be computed with the full vector $x_t$ in (5). This of course requires choosing a mask that is large enough; in the fMRI dataset, a subsampling factor of about $r = 10$ – that is only 10% of the entries of $x_t$ are observed – resulted in a similar $10 \times$ speed-up (see experimental section) to achieve the same accuracy as the original approach without subsampling. This point of view also justifies the natural scaling factor $\frac{\alpha}{\sqrt{p}}$ introduced in (11).

An efficient algorithm must address two challenges: (i) performing dictionary updates that do not depend on $p$ but only on the mask size; (ii) finding an approximate surrogate function that can be summarized by a few sufficient statistics. We provide a solution to these two issues in the next subsections and present the method in Algorithm 1.

2.1. Approximate Surrogate Function

To approximate the surrogate (8) from $\alpha_t$ computed in (13), we consider $h_t$ defined by

$$h_t(D) = \frac{1}{2} \text{Tr}(D^T D C_t - D^T B_t) + \frac{\lambda}{p} \sum_{i=1}^{t} \frac{s_i}{p} \Omega(\alpha_i)$$

(14)

with the same matrix $C_t$ as in (8), which is updated as

$$C_t \leftarrow \left(1 - \frac{1}{\ell} \right) C_{t-1} + \frac{1}{\ell} \alpha_t \alpha_t^\top,$$

(15)

and to replace $B_t$ in (8) by the matrix

$$B_t = \left( \sum_{i=1}^t M_i \right)^{-1} \sum_{i=1}^t M_i x_i \alpha_i^\top,$$

(16)

which is the same as (7) when $M_i = I$. Since $M_i$ is a diagonal matrix, $\sum_{i=1}^t M_i$ is also diagonal and simply “counts” how many times a row has been seen by the algorithm. $B_t$ thus behaves like $E[xx^\top]$ for large $t$, as in the fully-observed algorithm. By design, only rows of $B_t$ selected by the mask differ from $B_{t-1}$. The update can therefore be achieved in $O(s, k)$ operations:

$$B_t = B_{t-1} + \left( \sum_{i=1}^t M_i \right)^{-1} (M_i x_i \alpha_i^\top - M_t B_{t-1}).$$

(17)

This only requires keeping in memory the diagonal matrix $\sum_{i=1}^t M_i$, and updating the rows of $B_{t-1}$ selected by the mask. All operations only depend on the mask size $s_i$ instead of the signal size $p$.

2.2. Efficient Dictionary Update Rules

With a surrogate function in hand, we now describe how to update the codes $\alpha$ and the dictionary $D$ when only partial access to data is possible. The complexity for computing the sparse codes $\alpha_t$ is obviously independent from $p$ since (13) consists in solving a reduced penalized linear regression of $M_t x_t$ in $\mathbb{R}^{s_t}$ on $M_t D_{t-1}$ in $\mathbb{R}^{s_t \times k}$. Thus, we focus here on dictionary update rules.

The naive dictionary update (18) has complexity $O(kp)$ due to the matrix-vector multiplication for computing $\nabla_{d_j} g_t(D)$. Reducing the single iteration complexity of a factor $\frac{p}{2}$ requires reducing the dimensionality of the dictionary update phase. We propose two strategies to achieve that, both using block coordinate descent, by considering

$$d_j \leftarrow \text{Proj}_{\psi(t) \leq 1} \left[ d_j - \frac{1}{C_t[j,j]} M_t \nabla_{d_j} h_t(D) \right],$$

(18)

where $M_t \nabla_{d_j} h_t(D)$ is the partial derivative of $h_t$ with respect to the $j$-th column and rows selected by the mask.

Gradient step. The update (18) represents a classical block coordinate descent step involving particular blocks. Following Mairal et al. (2010), we perform one cycle over the columns warm-started on $D_{t-1}$. Formally, the gradient step without projection for the $j$-th component consists of updating the vector $d_j$

$$d_j \leftarrow d_j - \frac{1}{C_t[j,j]} M_t \nabla_{d_j} h_t(D)$$

$$= d_j - \frac{1}{C_t[j,j]} (M_t D c_j^t - M_t b_j^t),$$

(19)

where $c_j^t$, $b_j^t$ are the $j$-th columns of $C_t$, $B_t$ respectively. The update has complexity $O(k s_t)$ since it only involves $s_t$ rows of $D$ and only $s_t$ entries of $d_j$ have changed.

Projection step. Block coordinate descent algorithms require orthogonal projections onto the constraint set $\mathcal{C}$. In our case, this amounts to the projection step on the unit ball corresponding to the norm $\psi$ in (18). The complexity of such a projection is usually $O(p)$ both for $\ell_2$ and $\ell_1$-norms (see Duchi et al., 2008). We consider here two strategies.

Exact lazy projection for $\ell_2$. When $\psi = \ell_2$, it is possible to perform the projection implicitly with complexity $O(s_t)$. The computational trick is to notice that the projection amounts to a simple rescaling operation

$$d_j \leftarrow \frac{d_j}{\max(1, |d_j|_2)},$$

(20)

which may have low complexity if the dictionary elements $d_j$ are stored in memory as a product
Procedure 1 Dictionary Learning for Massive Data

Input: Initial dictionary: $D_0 \in \mathbb{R}^{p \times k}$, tolerance: $\epsilon$
$C_0 \leftarrow 0 \in \mathbb{R}^{k \times k}$;
$B_0 \leftarrow 0 \in \mathbb{R}^{p \times k}$;
$E_0 \leftarrow 0 \in \mathbb{R}^{p \times p}$ (diagonal);
$t \leftarrow 1$;
repeat
\[ \alpha_t \leftarrow \arg \min_{\alpha_t} \frac{1}{2} ||M_t(x_t - D_{t-1} \alpha_t)||_2^2 + \lambda \frac{Tr}{p} M_t \Omega(\alpha_t); \]
$E_t \leftarrow E_t + M_t;$
$A_t \leftarrow (1 - \frac{1}{t}) A_{t-1} + \frac{1}{t} \alpha_t \alpha_t^\top;$
$B_t \leftarrow B_{t-1} + E_t^{-1}(M_t x_t \alpha_t^\top - M_t B_{t-1});$
$D_t \leftarrow \text{dictionary update}(B_t, C_t, D_{t-1}, M_t);$
until $|h_t - h_{t-1}(D_{t-1})| - 1 | < \epsilon$
Output: $D$

d_j = f_j / \max(1, l_j)$ where $f_j$ is in $\mathbb{R}^p$ and $l_j$ is a rescaling coefficient such that $l_j = ||f_j||_2$. We code the gradient step (19) followed by $\ell_2$-ball projection by the updates
\[ n_j \leftarrow ||M_j f_j||_2^2 \]
\[ f_j \leftarrow f_j - \frac{\max(1, l_j)}{C_j[j,j]} (M_j D c_j - M_j b_j^j) \] (21)
\[ l_j \leftarrow \sqrt{l_j^2 - n_j + ||M_j f_j||_2^2} \]

Note that the update of $f_j$ corresponds to the gradient step without projection (19) which costs $O(k s_t)$, whereas the norm of $f_j$ is updated in $O(s_t)$ operations. The computational complexity is thus independent of $p$ and the only price to pay is to rescale the dictionary elements on the fly, each time we need access to them.

Exact lazy projection for $\ell_1$. The case of $\ell_1$ is slightly different but can be handled in a similar manner, by storing an additional scalar $l_j$ for each dictionary element $d_j$. More precisely, we store a vector $f_j$ in $\mathbb{R}^p$ such that $d_j = \text{Proj}_{\psi(\cdot) \leq 1} [f_j]$, and a classical result (see Duchi et al., 2008) states that there exists a scalar $l_j$ such that
\[ d_j = S_{\lambda}(f_j), \quad S_\lambda(u) = \text{sign}(u) \cdot \max(|u| - \lambda, 0) \] (22)

where $S_\lambda$ is the soft-thresholding operator, applied elementwise to the entries of $f_j$. Similar to the case $\ell_2$, the “lazy” projection consists of tracking the coefficient $l_j$ for each dictionary element and updating it after each gradient step, which only involves $s_t$ coefficients. For such sparse updates followed by a projection onto the $\ell_1$-ball, Duchi et al. (2008) proposed an algorithm to find the threshold $l_j$ in $O(s_t \log(p))$ operations. The lazy algorithm involves using particular data structures such as red-black trees and is not easy to implement; this motivated us to investigate another simple heuristic that also performs well in practice.

Approximate low-dimension projection. The heuristic consists in performing the projection by forcing the coefficients outside the mask not to change. This results in the orthogonal projection of each $d_j$ on $T_{\omega, t} = \{ d \text{ s.t. } \psi(d) \leq 1, (I - M_t) d = (I - M_t) d^{t-1} \}$, which is a subset of the original constraint set $\psi(\cdot) \leq 1$.

All the computations require only 4 matrices kept in memory $B$, $C$, $D$, $E$ with additional $F, L$ matrices and vectors for the exact projection case, as summarized in Alg. 1.

2.3. Discussion

Relation to classical matrix completion formulation. Our model is related to the classical $\ell_2$-penalized matrix completion model (e.g. Bell & Koren, 2007) we rewrite
\[ \sum_{i=1}^n ||M_i (x_i - D^\top \alpha_i)||_2^2 + \lambda \alpha_i ||2^2 + \lambda ||(\sum_{i=1}^n M_i)^2 D||_2^2 \] (23)

With quadratic regularization on $D$ and $A$ – that is, using $\Omega = ||.||_2^2$ and $\psi = ||.||_2$ – (11) only differs in that it uses a penalization on $D$ instead of a constraint. Srebro et al. (2004) introduced the trace-norm regularization to solve a convex problem equivalent to (23). The major difference is that we adopt a non-convex optimization strategy, thus losing the benefits of convexity, but gaining on the other hand the possibility of using stochastic optimization.

Practical considerations. Our algorithm can be slightly modified to use weights $w_i$ that differ from $\frac{1}{n}$ for $B$ and $C$, as advocated by Mairal (2013). It also proves beneficial to perform code computation on mini-batches of masked samples. Update of the dictionary is performed on the rows that are seen at least once in the masks $(M_t)_{batch}$.

3. Experiments

The proposed algorithm was designed to handle massive datasets: masking data enables streaming a sequence $(M_t x_t)_t$ instead of $(x_t)_t$, reducing single-iteration computational complexity and IO stress of a factor $r = \frac{B}{BCMT}$, while accessing an accurate description of the data. Hence,
Dictionary Learning for Massive Matrix Factorization

Figure 1. Acceleration of sparse matrix factorization with random subsampling on the HCP dataset (2TB). Reducing streamed data with stochastic masks permits $10 \times$ speed-ups without deteriorating goodness of fit on test data nor altering sparsity of final dictionary.

We analyze in detail how our algorithm improves performance for sparse decomposition of fMRI datasets. Moreover, as it relies on data masks, our algorithm is well suited for matrix completion, to reconstruct a data stream $(x_t)_t$ from the masked stream $(M_t x_t)_t$. We demonstrate the accuracy of our algorithm on explicit recommender systems and show considerable computational speed-ups compared to an efficient coordinate-descent based algorithm.

We use scikit-learn (Pedregosa et al., 2011) in experiments, and have released a python package\(^1\) for reproducibility.

3.1. Sparse Matrix Factorization for fMRI

Context. Matrix factorization has long been used on functional Magnetic Resonance Imaging (McKeown et al., 1998). Data are temporal series of 3D images of brain activity, to decompose in spatial modes capturing regions that activate together. The matrices to decompose are dense and highly redundant, both spatially and temporally: close voxels and successive records are correlated. Data can be huge: we use the whole HCP dataset (Van Essen et al., 2013), with $n = 2.4 \cdot 10^6$ (2000 records, 1 200 time points) and $p = 2 \cdot 10^5$, totaling 2 TB of dense data.

Interesting dictionaries for neuroimaging capture spatially-localized components, with a few brain regions. This can be obtained by enforcing sparsity on the dictionary: in our formalism, this is achieved with $\ell_1$-ball projection for $D$. We set $C = B_k^{\Omega}$, and $\Omega = \| \cdot \|_2^2$. Historically, such decomposition have been obtained with the classical dictionary learning objective on transposed data (Varoquaux et al., 2013): the code $\mathbf{A}$ holds sparse spatial maps and voxel time-series are streamed. However, given the size of $n$ for our dataset, this method is not usable in practice.

Handling such volume of data sets new constraints. First, efficient disk access becomes critical for speed. In our case, learning the dictionary is done by accessing the data in row batches, which is coherent with fMRI data storage: no time is lost seeking data on disk. Second, reducing IO load on the storage is also crucial, as it lifts bottlenecks that appear when many processes access the same storage at the same time, e.g. during cross-validation on $\lambda$ within a supervised pipeline. Our approach reduces disk usage by a factor $r$. Finally, parallel methods based on message passing, such as asynchronous coordinate descent, are unlikely to be efficient given the network / disk bandwidth that each process requires to load data. This makes it crucial to design efficient sequential algorithms.

Experiment We quantify the effect of random subsampling for sparse matrix factorization, in term of speed and accuracy. A natural performance evaluation is to measure an empirical estimate of the loss $l$ defined in Eq. 4 from unseen data, to rule out any overfitting effect. For this, we evaluate $l$ on a test set $(x_t)_{t<N}$. Practically, we sample $(x_t)_t$ in a pseudo-random manner: we randomly select a record, from where we select a random batch of rows $x_t$ – we use a batch size of 40, empirically found to be efficient. We load $M_t x_t$ in memory and perform an iteration of the algorithm. The mask sequence is sampled by breaking random permutation vectors into chunks of size $p/r$.

Results Fig. 1(a) compares our algorithm with subsampling ratios $r \in \{4, 8, 12\}$ to vanilla online dictionary learning algorithm ($r = 1$), plotting trajectories of the test objective against real CPU time. There is no obvious choice of $\lambda$ due to the unsupervised nature of the problem: we use $10^{-3}$ and $10^{-4}$, that bounds the range of $\lambda$ providing interpretable dictionaries.

First, we observe the convergence of the objective function for all tested $r$, providing evidence that the approximations made in the derivation of update rules does not break convergence for such $r$. Fig. 1(b) shows the validity of the obtained dictionary relative to the reference output: both objective function and $\ell_1/\ell_2$ ratio – the relevant value to measure sparsity in our setting – are comparable to the baseline values, up to $r = 8$. For high regularization and $r = 12$, our algorithm tends to yield somewhat sparser solutions (5% lower $\ell_1/\ell_2$) than the original algorithm, due to the approximate $\ell_1$-projection we perform. Obtained maps

\(^1\)http://github.com/arthurmensch/modl
still proves as interpretable as with baseline algorithm.

Our algorithm proves much faster than the original one in finding a good dictionary. Single iteration time is indeed reduced by a factor \( r \), which enables our algorithm to go over a single epoch \( r \) times faster than the vanilla algorithm and capture the variability of the dataset earlier. To quantify speed-ups, we plot the empirical objective value of \( D \) against the number of observed records in Fig. 3. For \( r \leq 12 \), increasing \( r \) little reduces convergence speed per epoch: random subsampling does not shrink much the quantity of information learned at each iteration.

This brings a near \( \times r \) speed-up factor: for high and low regularization respectively, our algorithm converges in 3 and 10 hours with subsampling factor \( r = 12 \), whereas the vanilla online algorithm requires about 30 and 100 hours. Qualitatively, Fig. 2 shows that with the same time budget, the proposed reduction approach with \( r = 12 \) on half of the data gives better results than processing a small fraction of the data without reduction: segmented regions are less noisy and closer to processing the full data.

These results advocate the use of a subsampling rate of \( r \approx 10 \) in this setting. When sparse matrix decomposition is part of a supervised pipeline with scoring capabilities, it is possible to find \( r \) efficiently: start by setting it reasonably high and decrease it geometrically until supervised performance (e.g. in classification) ceases to improve.

### 3.2. Collaborative Filtering with Missing Data

We validate the performance of the proposed algorithm on recommender systems for explicit feedback, a well-studied matrix completion problem. We evaluate the scalability of our method on datasets of different dimension: MovieLens 1M, MovieLens 10M, and 140M ratings Netflix dataset.

We compare our algorithm to a coordinate-descent based method (Yu et al., 2012), that provides state-of-the-art convergence time performance on our largest dataset. Although stochastic gradient descent methods for matrix factorization can provide slightly better single-run performance (Takács et al., 2009), these are notoriously hard to tune and require a precise grid search to uncover a working schedule of learning rates. In contrast, coordinate descent methods do not require any hyper-parameter setting and are therefore more efficient in practice. We benchmarked various recommender-system codes (MyMediaLite, LibFM, SoftImpute, spira\(^2\)), and chose coordinate descent algorithm from spira as it was by far the fastest.

**Completion from dictionary \( D_t \).** We stream user ratings to our algorithm: \( p \) is the number of movies and \( n \) is the number of users. As \( n \gg p \) on Netflix dataset, this increases the benefit of using an online method. We have observed comparable prediction performance streaming item ratings. Past the first epoch, at iteration \( t \), every column \( i \) of \( X \) can be predicted by the last code \( \alpha_{i,(i,t)} \) that was computed from this column at iteration \( t-1 \). At iteration \( t \), for all \( i \leq |n| \), \( x_{i}^{\text{pred}} = D_{i} \alpha_{i,(i,t)} \). Prediction thus only requires an additional matrix computation after the factorization.

**Preprocessing.** Successful prediction should take into account user and item biases. We compute these biases on train data following Hastie et al. (2014) (alternated de-
Table 1. Comparison of performance and convergence time for online masked matrix factorization and coordinate descent method. Convergence time: score is under 0.1% deviation from final root mean squared error on test set – 5 runs average. CD: coordinate descent; MODL: masked online dictionary learning.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Test RMSE</th>
<th>Convergence time</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CD</td>
<td>MODL</td>
<td></td>
</tr>
<tr>
<td>ML 1M</td>
<td>0.872</td>
<td>0.866</td>
<td>6 s</td>
</tr>
<tr>
<td>ML 10M</td>
<td>0.802</td>
<td>0.799</td>
<td>223 s</td>
</tr>
<tr>
<td>NF (140M)</td>
<td>0.938</td>
<td>0.934</td>
<td>1714 s</td>
</tr>
</tbody>
</table>

We use them to center the samples \((x_t)_t\) that are streamed to the algorithm, and to perform final prediction.

Tools and experiments. Both baseline and proposed algorithm are implemented in a computationally optimal way, enabling fair comparison based on CPU time. Benchmarks were run using a single 2.7 GHz Xeon CPU, with a 30 components dictionary. For Movielens datasets, we use a random 25% of data for test and the rest for training. We average results on five train/test split for MovieLens in Table 1. On Netflix, the probe dataset is used for testing. Regularization parameter \(\lambda\) is set by cross-validation on the training set: the training data is split 3 times, keeping 33% of Movielens datasets for evaluation and 1% for Netflix, and grid search is performed on 15 values of \(\lambda\) between \(10^{-2}\) and 10. We assess the quality of obtained decomposition by measuring the root mean square error (RMSE) between prediction on the test set and ground truth. We use mini-batches of size \(\frac{n}{m}\).

Results. We report the evolution of test RMSE along time in Fig. 4, along with its value at convergence and numerical convergence time in Table 1. Benchmarks are performed on the final run, after selection of parameter \(\lambda\).

The two variants of the proposed method converge toward a solution that is at least as good as that of coordinate descent, and slightly better on Movielens 10M and Netflix. Our algorithm brings a substantial performance improvement on medium and large scale datasets. On Netflix, convergence is almost reached in 4 minutes (score under 0.1% deviation from final RMSE), which makes our method 6.8 times faster than coordinate descent. Moreover, the relative performance of our algorithm increases with dataset size. Indeed, as datasets grow, less epochs are needed for our algorithm to reach convergence (Fig. 4). This is a significant advantage over coordinate descent, that requires a stable number of cycle on coordinates to reach convergence, regardless of dataset size. The algorithm with partial projection performs slightly better. This can be explained by the extra regularization on \((D_t)_t\) brought by this heuristic.

Learning weights. Unlike SGD, and similar to the vanilla online dictionary learning algorithm, our method does not critically suffer from hyper-parameter tuning. We tried weights \(w_t = \frac{1}{t^\beta}\) as described in Sec. 2.3, and observed that a range of \(\beta\) yields fast convergence. Theoretically, Mairal (2013) shows that stochastic majorization-minimization converges when \(\beta \in (.75, 1]\). We verify this empirically, and obtain optimal convergence speed for \(\beta \in [.85, .95]\). (Fig. 5). We report results for \(\beta = 0.9\).

Conclusion

Whether it is sensor data, as fMRI, or e-commerce databases, sample sizes and number of features are rapidly growing, rendering current matrix factorization approaches intractable. We have introduced an online algorithm that leverages random feature subsampling, giving up to 8-fold speed and memory gains on large data. Datasets are getting bigger, and they often come with more redundancies. Such approaches blending online and randomized methods will yield even larger speed-ups on next-generation data.
Acknowledgements

The research leading to these results was supported by the ANR (MACARON project, ANR-14-CE23-0003-01 – NiConnect project, ANR-11-BINF-0004NiConnect) and has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 604102 (HBP).

References


