Sketched Stochastic Dictionary Learning for large-scale data and application to high-throughput mass spectrometry

Olga Permiakova, Thomas Burger

Sketched Stochastic Dictionary Learning for large-scale data and application to high-throughput mass spectrometry

Olga Permiakova, Thomas Burger

Univ. Grenoble Alpes, CNRS, CEA, Inserm, ProFI FR2048,
Grenoble, France
thomas.burger@cea.fr

Abstract

Factorization of large data corpora has emerged as an essential technique to extract dictionaries (sets of patterns that are meaningful for sparse encoding). Following this line, we present a novel algorithm based on compressive learning theory. In this framework, the (arbitrarily large) dataset of interest is replaced by a fixed-size sketch resulting from a random sampling of the data distribution characteristic function. We apply our algorithm to the extraction of chromatographic elution profiles in mass spectrometry data, where it demonstrates its efficiency and interest compared to other related algorithms.

Keywords: Dictionary learning; Stochastic gradient descent; Compressive statistical learning; Nesterov accelerated gradient descent; Computational mass spectrometry; Matrix factorization

1 Introduction

Finding a linear decomposition of an observed signal $x \in \mathbb{R}^s$ is essential for many applications, as it provides a way to exhibit its elementary constitutive patterns, as well as to denoise it. Formally, this task amounts to finding a vector of coefficients $c = (c_1, \ldots, c_K) \in \mathbb{R}^K$, referred to as *code*, such that:

$$x = c_1 \cdot d_1 + \cdots + c_K \cdot d_K + \epsilon, \tag{1}$$

where $D = \{d_1, \ldots, d_K\} \in \mathbb{R}^{s \times K}$ is a matrix referred to as *dictionary*, composed of K s-dimensional column vectors (the *dictionary atoms*), and

where ϵ represents the (hopefully small) part of x that is not explained by D. While many solutions to this problem are already available when D is known, the decomposition of a signal, which potential constitutive elementary patterns are unknown (referred to as blind source separation), is much more difficult. As a result, despite being almost 30 years old [1], this problem still focuses investigations.

According to compressive sensing theory [2], a good dictionary is such that any signal can be precisely approximated using few dictionary atoms only, *i.e.*, only a restricted number of c_k are non-null in Decomposition (1); and the fewer the better. As emphasized in [3], this type of representations, referred to as *sparse representations*, are widespread in many real-life applications: image denoising [4], super resolution [5], compression [6], etc.

The oldest strategies to decompose signals have used Riesz bases as dictionaries (*e.g.*, Fourier, wavelet or curvelet bases [7]). Their mathematical properties have made the decomposition straightforward, yet, for highly complex real-life signals, sparse representations have generally been achievable only at the price of an important unexplained residue ϵ. In fact, it has since then been established [8] that sparse representations were easier to obtain when Decomposition (1) involved an overcomplete dictionary, *i.e.* a dictionary which size exceeds the signal dimensionality $K > s$. However, finding an overcomplete dictionary yields two (related) difficulties: First, the corresponding matrix D is not full-rank, which can potentially lead to numerical issues. Second, Decomposition (1) may not be unique, so that additional constraints are usually necessary to lead to a well-posed problem and a practically satisfactory solution. To overcome them, it has been proposed to extract elementary patterns from a set of signals akin to that for which a decomposition is sought, and to form an overcomplete dictionary with these patterns. This approach, referred to as *dictionary learning*, has been demonstrated to lead to dictionaries that are of real practical interest, for three reasons: First, they capture well the specificities of the data [9]. Second, they allow for even sparser representations. Third, their atoms are easier to relate to physical signals and thus to interpret [10]. Concretely, learning a dictionary from a set of observed signals $X \in \mathbb{R}^{s \times N}$, is related to finding a decomposition of X into a product of two low rank matrices [11]. The effectiveness of this matrix factorization approach has been illustrated in many applications, such as medical signal modeling and analysis [10], natural image processing [12], audio and video processing [13]. In this article, we aim to apply dictionary learning to another type of data: those resulting from the high-throughput mass spectrometry analysis of complex biological samples.

Mass spectrometry (MS) coupled with liquid chromatography (LC) is a commonly used analytical chemistry technique, which has witnessed an increasing popularity in the last decade [14], due to its application to omics biology. Notably, it has become the method of choice for proteome, metabolome and lipidome investigations. Despite increasing resolution and cycle speed, the LC-MS pipeline is still challenged by the complexity of classical biological samples: Concretely, the number of biomolecules to identify and quantify remains larger than the measurement capabilities. To increase the sample coverage, relying on multiplexed measurements [15] (that is, recording a single signal for several biomolecules which intensities are summed up, see Figure 1a) has become increasingly popular. However, interpreting the resulting data requires a subsequent demultiplexing step which intuitively translates (see Figure 1b) into solving a blind source separation problem [16]. To tackle it, we follow the line of [17], which proposes to denoise simple LC-MS data by relying on a matrix factorization formulation. Practically, the LC-MS data can easily be formatted into a matrix: Broadly, the LC can be seen as a way to serialize the analytes into the MS, so as to avoid that too many of them are concomitantly analyzed. Thus, if the mass spectra produced over time are stored as vectors of high dimensionality ($N \approx 10^5$) and stacked as matrix rows, the matrix columns can be interpreted as chromatograms (a.k.a. elution profiles, see Figure 1c). A chromatogram is a vector that represents an analyte's flow rate outputted from the LC toward the MS. This flow rate being a physical signal discretized at a relatively high frequency (which explains why s lies between 10^3 and 10^4) its smoothness can be leveraged. Notably, we have formerly established [18] it can be used to extract meaningful chemical patterns through cluster analysis. In this work, we propose to learn a dictionary of chromatograms to subsequently improve the decomposition of LC-MS data, with a sparsity level related to the number of biomolecules which measurements have been multiplexed.

Extracting a meaningful dictionary from LC-MS data comes along with numerous challenges. First, the dictionary atoms must be interpretable as chromatograms (*i.e.* smooth, non-negative, slightly heavy right-tailed waveforms, see Figure 1). Second, owing to the number of analytes in a classical sample (up to tens of thousands), which largely exceeds the chromatographic signal dimensionality, the dictionary must be highly overcomplete. Third, LC-MS data are already rather big and their size is ever-increasing due to the constant improvement of MS resolution and cycle speed. Therefore, processing them in a time compliant with the various constraints of a standard analytical platform is a computational challenge that requires scalable solutions.

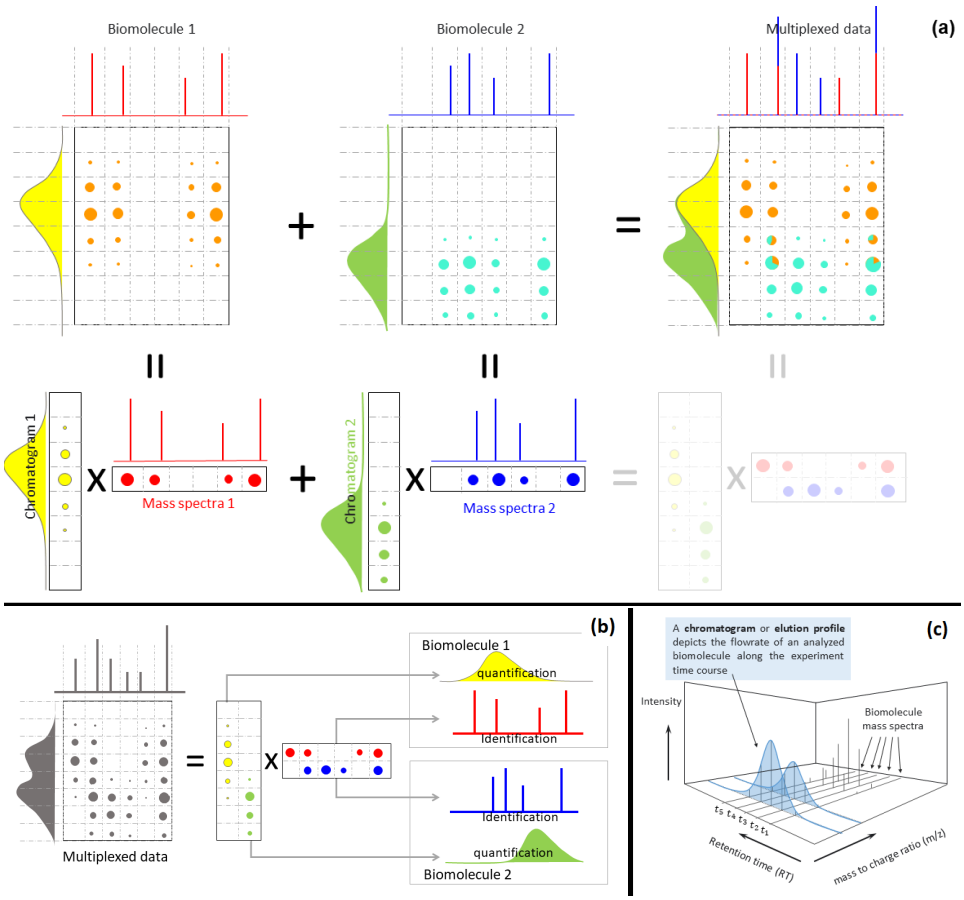

Figure 1: The multiplexing process in LC-MS data measurements is additive (a) so that the demultiplexing operation amounts to a matrix factorization (b) as LC-MS data can be easily formatted into a matrix (c).

We hereby describe a new approach meeting these constraints. From a methodological viewpoint, our contribution essentially lies in an original formulation of the dictionary learning problem which leverages recent developments in compressive learning. This framework aims to design algorithms which operate on a so-called data sketch in place of the original input data (which can be disposed of once the sketch has been computed). The main interest of this methodology is to enable data compression (making the sketch coarser or finer) according to the desired trade-off between accuracy and computational load. Adapting this framework to improve dictionary learning yields a difficulty: computing the codes for each data point requires having access to the entire input data. As this intrinsically limit the computational gain of the sketching procedure, we propose to rely on a stochastic gradient descent algorithm to minimize the dictionary learning objective function. In spite of this, relying on a data sketch to extract dictionary atoms remains particularly appealing. As both tasks aim at extracting salient patterns from the data, the later can leverage the distributions captured by the former. A combination of above ideas yields a new method referred to as Sketched Stochastic Dictionary Learning (SSDL). According to our experimental validation, this method improves the state-of-the-art to extract a set of meaningful patterns from LC-MS data with a small computational footprint.

The article is structured as follows: Section 2 gathers the related works, including standard dictionary learning formulation, presentation of state-of-the-art methods and summary of background knowledge that are instrumental to a clear exposure of SSDL. Then Section 3 presents SSDL. Finally, Section 4 is dedicated to experimental validations on LC-MS data.

2 Related works

2.1 Notations

In the rest of the article, matrices are noted with capital letters like A or X and vectors with lowercase ones, like a or x (however, both types of letters may also be used for integers, sets, etc.). The j^{th} column (respectively, row) of matrix X is denoted $X_{:,j}$ (respectively, $X_{j,:}$), or possibly x_j if the corresponding vector has been previously defined. Following the same line, the $(j,k)^{\text{th}}$ element of X is denoted $X_{j,k}$. The pseudo-inverse of matrix A is denoted as $A^\dagger$ and its transpose is $A^\top$. On the other hand, A^t denotes the t^{th} iteration of the computation of A. By abuse of notation, $t+\frac{1}{2}$ is used to refer to an intermediate step of iteration t. The imaginary number

$(\sqrt{-1})$ is denoted as i and $\bar{z} \in \mathbb{C}$ is the complex conjugate of $z \in \mathbb{C}$, *i.e.*, $z = \mathrm{Re}(z) + i \cdot \mathrm{Im}(z)$ and $\bar{z} = \mathrm{Re}(z) - i \cdot \mathrm{Im}(z)$. Finally, $||.||_p$ is the L^p norm, $[\ .\ ]_{j=1}^{m}$ represents the construction of a vector from its m components, ∇ is the gradient operator and $\mathbb{E}$ is expectation one.

2.2 Classical dictionary learning strategies

Let $X = \{x_1, \ldots, x_N\} \in \mathbb{R}^{s \times N}$ be a data matrix. The classical formulation of dictionary learning reads as the following joint optimization problem:

$$\min_{\substack{C \in \mathbb{R}^{K \times N} \\ D \in S}} \frac{1}{N} \sum_{k=1}^{N} \|x_k - c_k \cdot D\|_2^2 + \lambda \cdot \|c_k\|_1 \tag{2}$$

where C is a matrix gathering all code vectors c_k; where λ is the regularization parameter of a LASSO penalty (Least Absolute Shrinkage and Selection Operator, [19]), which controls the representation sparsity; and where S is the convex set where dictionary atoms can be picked up. It has been proven in [20] that solving the minimization problem of Eq. (2) leads to sparse representations. Moreover, this problem being convex with respect to C and D separately, most methods to solve it rely on an alternative minimization scheme (*i.e.*, minimization of Eq. (2) with respect to one variable while the other is fixed):

$$\min_{C \in \mathbb{R}^{K \times N}} \frac{1}{N} \sum_{k=1}^{N} \|x_k - c_k \cdot D\|_2^2 + \lambda \cdot \|c_k\|_1 \tag{3}$$

$$\min_{D \in S} \frac{1}{N} \sum_{k=1}^{N} \|x_k - c_k \cdot D\|_2^2 \tag{4}$$

The first sub-problem (Eq. 3), which consists in computing the code matrix for a given dictionary, is referred to as *sparse coding*. It can be solved using LASSO [19], LARS (least-angle regression, [21]), iterative shrinkage thresholding [22] or other numerical schemes relying on descent paradigms [23, 24]. However, the specificity of any dictionary learning approach essentially lies in the technique used to solve Eq. (4), *i.e.*, the *dictionary update*. For example, Engan et al. [25] rely on an analytical solution to re-compute the dictionary at each iteration: $D = X \cdot C^{\dagger}$, where $C^{\dagger}$ is the pseudo-inverse of the code matrix. Alternatively, K-SVD [26] updates each dictionary atoms independently by performing singular value decomposition. Finally, dictionary learning can also be tackled in the Bayesian [27, 28, 29] or optimal transport [30] frameworks.

2.3 Large-scale dictionary learning techniques

Stochastic or online learning [31, 32] is an efficient and broadly used technique to train a dictionary from a large dataset. It consists in updating the dictionary and calculating the code at each iteration of the minimization procedure, by relying on a randomly selected subset of signals only (possibly, a single one). As working on a small subset drastically reduces the computational time of both sub-problems, multiple passes through the data (called *epochs*) can be used until convergence. We have singled out two online dictionary learning methods to benchmark against our approach: MODL (Massive Online Dictionary Learning, [33, 34]) and IcTKM (Iterative Compressed-Thresholding and K-Means, [35]). The former is the state-of-the-art approach, and it demonstrates excellent computational performance on extremely large datasets. The latter is a recently published method which underlying mathematics are conceptually related to those leveraged in the present work, making the comparison worth of interest. More precisely, MODL minimizes Objective Functions (3) and (4): Sparse coding is achieved using LARS algorithm, while coordinate gradient descent is applied to update dictionary atoms. Formally, at the t^{th} iteration, the dictionary atoms are recomputed as follows:

$$d_k^t = d_k^{t-1} - \frac{1}{A_{kk}^t} \cdot \left(D^{t-1} \cdot A_{:,k}^t - B_{:,k}^t \right), \tag{5}$$

where auxiliary matrices A^t and B^t are defined as $A^t = \frac{1}{t} \sum_{k=1}^{t} c_k \cdot c_k^\top \in \mathbb{R}^{K \times K}$ and $B^t = \frac{1}{t} \sum_{k=1}^{t} x_k \cdot c_k^\top \in \mathbb{R}^{s \times K}$; and where $A_{:,k}^t$ and $B_{:,k}^t$ denote the k^{th} columns of matrices A^t and B^t respectively. Using these auxiliary matrices allows to gather the statistics about the signals $x_1, \ldots, x_t$ and codes $c_1, \ldots, c_t$ observed at previous iterations without explicitly storing them in memory. This strategy provides low memory consumption and computational cost, at least for small K. MODL is compliant with positiveness constraints on both the dictionary and code matrices, as required by LC-MS data (see Section 1). Finally, MODL embeds an optional dimensionality reduction step based on random projections [36].

Although IcTKM reformulates dictionary learning as a constrained minimization problem, it also proposes a solution based on alternate minimization. Instead of a regularization parameter, the sparsity level Λ_{IcTKM} is directly defined through a constraint on the number of non-zero elements in the columns of the codes. Sparse coding is solved using Iterative Thresholding [37]. The dictionary update is carried out by computing the K-residual means (following the known equivalence between clustering and matrix fac-

torization [38]). Concretely, each dictionary atom d_k is updated by averaging the data vectors x_j with non-null code coefficients C_{kj}. To speed up the computations, IcTKM relies on fast Johnson–Lindenstrauss transform [39].

2.4 Scaling-up by sketching

The method proposed in this article borrows two important features from the large-scale machine learning literature and adapt them to the dictionary learning context. The first one is the *compressive statistical learning* framework [40]. Its seminal idea is to summarize the data collection into a complex vector of fixed size, referred to as the *data sketch*, so that the algorithm complexity does not depend on the data size anymore. Concretely, the data sketch is constructed by sampling the characteristic function of the empirical data distribution $P(X)$:

$$SK(X) = \left[\mathbb{E}_{x\sim P(X)}\left(e^{iw_j^\top x}\right)\right]_{j=1}^{m}, \tag{6}$$

where $w_j \in \mathbb{R}^s$ is one of m vectors of frequencies (m being an application-dependent parameter). The w_j's are randomly sampled from some predefined distribution; usually, a normal distribution with null mean and standard deviation estimated on the Fourier transform of a data subset, see [40]. Starting from this theoretical ground, the main challenge is to adapt the machine learning method of interest so that it operates on the data sketch rather than on the original data. The authoring team has demonstrated both the practical interest and the efficiency of this approach on various problems, including Gaussian mixture estimation [40] and data clustering [41]. However, to the best of our knowledge, there is no dictionary learning method based on this framework.

Our second cornerstone is Nesterov accelerated gradient descent method (NAGD, see [42] as well as its more recent formulation [43]). Conceptually, it is akin to classical gradient descent, however, it includes an additional term, the *momentum*, denoted as η (a weighted average of the gradient vectors computed in the previous iterations). Adding this momentum term makes quadratic convergence possible in the deterministic cases [42]. The NAGD update rule reads:

$$\begin{aligned} D^{t+\frac{1}{2}} &= D^t + \alpha\cdot\eta^t \\ \eta^{t+1} &= \alpha\cdot\eta^t - \gamma\cdot\nabla_D f\left(D^{t+\frac{1}{2}}\right) \\ D^{t+1} &= D^t + \eta^{t+1} \end{aligned} \tag{7}$$

where f is the function to minimize; where α is the momentum weight; where γ defines the length of each gradient step (the *learning rate*); and where $D^{t+\frac{1}{2}}$ is referred to as the *ahead*. Recently, NAGD scheme has attracted great interest for stochastic optimization: Its convergence under convex and smooth optimization has been heavily documented [44, 45], but scarce results are so far available for non-convex cases. Despite, it is of practical interest, as when correctly tuned, it outperforms the classical stochastic method [31].

3 SSDL method

3.1 Objective function

As SSDL follows a classical alternate minimization scheme, the dictionary update and the code computation objective functions can be separated. The former differs from the classical dictionary update (Eq. 4), as we propose to include the sketching operator of Eq. (6). As for the code computations, we have modified Eq. (3) to fit the stochastic learning framework. Concretely, at each iteration, the code matrix is computed only for a data subset selected uniformly at random, denoted $\hat{X} = \{\hat{x}_1, \ldots, \hat{x}_n\} \in \mathbb{R}^{s \times n}$, where $n < N$:

$$C^* = \arg \min_{C \in \mathbb{R}^{K \times n}} \frac{1}{n} \sum_{k=1}^{n} \|\hat{x_k} - D \cdot c_k\|_2^2 + \lambda \cdot \|c_k\|_1, \tag{8}$$

Then SSDL looks for a dictionary D such that the data sketch $SK(X)$ is as close as possible to the sketch of the decomposition $SK(D \cdot C^*)$:

$$\min_{D \in S} F(D, C^*), \text{ with } F(D, C^*) = \|SK(X) - SK(D \cdot C^*)\|_2^2 \tag{9}$$

Since the sketching procedure amounts to sampling an empirical characteristic function, D does not only represent the observed data, but to some extent, its underlying distribution. Therefore, the dictionary resulting from this procedure can be expected to generalize well to other data with similar distribution. This behavior should moreover be strengthened by an adequate tuning of the frequency generation procedure [40] as it aims to capture only the most relevant features and eliminate noise. Finally, it is also of interest in a stochastic learning context: despite the use of a random sampling procedure on the training set, the optimizer can also rely on the complete data summary provided by the data sketch.

These changes in the objective functions lead to several advantages: First, the stochastic minimization is fully efficient, as the additional computations it requires are immaterial on small data batches. Second, the

gradient of Eq. (9) does not contain the term $D \cdot A$, where the computational footprint of matrix $A = \sum_{k=1}^{n} c_k \cdot c_k^\top \in \mathbb{R}^{K \times K}$ could be important for highly overcomplete dictionary. Finally, the data sketch $SK(X)$ is computed once, at the algorithm initialization (afterwards the data sketch remains unchanged) and this initial computation parallelizes well in case of extremely large datasets.

3.2 Minimization

At each iteration of SSDL, the dictionary is updated by performing one gradient step as in Nesterov scheme (7), with the gradient of Objective function (9) reading:

$$\begin{aligned} \nabla_{d_\ell} \|SK(X) - SK(D \cdot C^*)\|^2 &= \nabla_{d_\ell} \left(z \cdot z^\top \right) \\ &= 2 \left(\nabla_{d_\ell} \mathrm{Re}(z)^\top \cdot \mathrm{Re}(z) + \nabla_{d_\ell} \mathrm{Im}(z)^\top \cdot \mathrm{Im}(z) \right) \\ &= 2 \cdot \nabla_{d_\ell} q^\top \cdot q \in \mathbb{R}^s, \end{aligned} \tag{10}$$

where $z = SK(X) - SK(D \cdot C^*) \in \mathbb{C}^m$, $q = [\mathrm{Re}(z), \mathrm{Im}(z)] \in \mathbb{R}^{2m}$ is a vector obtained by stacking the real and imaginary parts of the complex vector z and

$$\nabla_{d_\ell} q^\top = \left[\frac{\delta}{\delta d_\ell} \mathrm{Re}(z)^1, \ldots, \frac{\delta}{\delta d_\ell} \mathrm{Re}(z)^m, \frac{\delta}{\delta d_\ell} \mathrm{Im}(z)^1, \ldots, \frac{\delta}{\delta d_\ell} \mathrm{Im}(z)^m \right] \in \mathbb{R}^{s \times 2m} \tag{11}$$

with

$$\frac{\delta}{\delta d_\ell} \mathrm{Re}(z)^j = -\frac{\delta}{\delta d_\ell} \mathrm{Re}(SK^j(D \cdot C^*)) = \frac{1}{n} \sum_{k=1}^{n} C^*_{\ell k} \cdot \sin \left(w_j^\top \cdot D \cdot C^*_{:,k} \right) \cdot w_j \tag{12a}$$

$$\frac{\delta}{\delta d_\ell} \mathrm{Im}(z)^j = -\frac{\delta}{\delta d_\ell} \mathrm{Im}(SK^j(D \cdot C^*)) = -\frac{1}{n} \sum_{k=1}^{n} C^*_{\ell k} \cdot \cos \left(w_j^\top \cdot D \cdot C^*_{:,k} \right) \cdot w_j \tag{12b}$$

and with SK^j (respectively, $\mathrm{Re}(z)^j$, $\mathrm{Im}(z)^j$) being the j^{th} coordinate of the sketch vector (respectively the real and imaginary part vectors).

However, the dictionary update of SSDL differs from the general Nesterov scheme. First, to ensure the positivity of the dictionary atoms, the

Algorithm 1 Sketched Stochastic Dictionary Learning

Input: Data matrix $X \in \mathbb{R}^{s \times N}$; Initial dictionary D^0; Initial learning rate γ_0; Decay parameter ν; Momentum weight α; Batch size n; Regularization weight λ.

1: **Initialization:**
2: Compute $SK(X)$ from X following [40]
3: $t = 0$, $\eta^0 = 0$
4: **Repeat until** convergence:
5: Construct a data batch $\hat{X}^t \in \mathbb{R}^{s \times n}$.
6: Sparse coding: $C^t = \min_{C \in \mathbb{R}^{K \times n}} \frac{1}{n} \sum_{k=1}^{n} \|\hat{x_k}^t - D^t \cdot c_k\|_2^2 + \lambda \cdot \|c_k\|_1$
7: Dictionary update:
8: $D^{t+\frac{1}{2}} = \max(D^t + \alpha \cdot \eta^t, 0)$
9: $\gamma_t = \frac{\gamma_0}{(1+(t-1)\cdot\nu)}$
10: $\eta^{t+1} = \alpha \cdot \eta^t - \gamma_t \cdot \nabla_D F(D^{t+\frac{1}{2}}, C^t)$
11: $D^{t+1} = \mathcal{P}_S\left(D^t + \eta^{t+1}\right)$, where $\mathcal{P}_S(\cdot)$ is a Euclidean projection on convex set $S = \{d \in \mathbb{R}_+^K \mid \|d\|_2 \leq 1\}$

ahead dictionary is projected on $\mathbb{R}_+^{s \times K}$ (see Alg. 1, Line 8). Second, SSDL uses a decaying learning rate (see Alg. 1, Line 9):

$$\gamma_t = \frac{\gamma_0}{(1 + (t-1) \cdot \nu)}. \tag{13}$$

This is motivated by the following fact: using a large γ far away from the minimum and progressively decreasing it allows to accelerate the convergence. As a drawback, it requires to tune an additional parameter, ν (the decay speed). The last difference relies in the Euclidean projection on $S = \{d \in \mathbb{R}_+^K \mid \|d\|_2^2 \leq 1\}$, which is included after the dictionary update (see Alg. 1, Line 11) to avoid too small scalars in the code matrix C. The complete pseudocode of SSDL is presented in Alg. 1.

3.3 Implementation and complexity

SSDL is implemented in R language. The gradient vector calculation is implemented in C++ and it is wrapped to R using the `Rcpp` package [46] and parallelized using the `RcppParallel` package [47]. The sparse coding problem is addressed by the `GLMNET` R function from the `glmnet` package [48], which is parallelized using the `mclapply` R function from the `parallel` package [49]. The File-backed Big Matrix (`FBM`) class of the

`bigstatsr` [50] package is used to store and to manipulate the matrices that are too large to be memory allocated. The SSDL code is made available on gitlab https://gitlab.com/Olga.Permiakova/ssdl as well as on the CRAN (Comprehensive R Archive Network): https://cran.r-project.org/web/packages/SSDL/index.html.

A global theoretical complexity algorithm would be fragile, as each compared method does not necessarily need the same number of iterations to converge. However, it is possible to compare the computational complexity of a single iteration. Concretely, the two steps of IcTKM have the following complexity:

- Compressed thresholding: $\mathcal{O}\left(\frac{s}{r_{IcTKM}}KN\right)$,
- Projections in the dictionary update: $\mathcal{O}\left(N\Lambda_{IcTKM}^3\right)$,

where r_{IcTKM} is the so-called reduction parameter (see Section 4.3). Contrarily to IcTKM, for which each iteration traverses the entire input data, MODL and SSDL process a single batch per iteration. The complexity of MODL reads:

- Gram matrix computation and dictionary update by coordinate descent: $\mathcal{O}\left(\frac{s}{r_{MODL}}K^2\right)$,
- Auxiliary matrices computation: $\mathcal{O}\left(\frac{s}{r_{MODL}}Kn_{MODL}\right)$,
- Code computation: $\mathcal{O}\left(Kn_{MODL}\tau_{MODL}^2\right)$,

where τ_{MODL} is the sparsity level resulting from tuning regularization parameter λ_{MODL}; r_{MODL} is MODL reduction parameter; and n_{MODL} is MODL batch size (see Section 4.3). Finally, SSDL complexity reads:

- Sparse coding: $\mathcal{O}\left(sKn\right)$,
- Dictionary update: $\mathcal{O}\left(mKs\right) + \mathcal{O}\left(mKn\right)$,
- Euclidean projection: $\mathcal{O}\left(sK\right)$.

However, it should be noted that the computational complexity of a single SSDL iteration does not account for the initial data sketching, which complexity is $\mathcal{O}(msN)$. As a result, one expects SSDL to be efficient only when the data are large enough to require a sufficient amount of batch-wise iterations (see Section 4.4). Finally, the total complexity per iteration are summarized in Table 1.

Algorithm	Single iteration complexity
IcTKM	$\mathcal{O}\left(\frac{s}{r_{IcTKM}}KN\right) + \mathcal{O}\left(N\Lambda_{IcTKM}^3\right)$
MODL	$\mathcal{O}\left(\left[\frac{s}{r_{MODL}} + \tau_{MODL}^2\right] Kn_{MODL} + \frac{s}{r_{MODL}}K^2\right)$
SSDL	$\mathcal{O}\left(\left[s+m\right] Kn + mKs\right)$

Table 1: Summary of the IcTKM, MODL and SSDL complexities for a single iteration.

4 Experimental validation

4.1 Methodology of evaluation

The validation of SSDL relies on a proteomic dataset hereafter referred to as the Ecoli dataset. It has resulted from the LC-MS analysis of a sample of *Escherichia coli* bacteria, see Section 4.2, as well as [18] for a more comprehensive description. SSDL capabilities to extract a dictionary from the Ecoli dataset are compared to those of MODL and IcTKM, both in terms of execution time and of dictionary atom quality. The machine used to benchmark SSDL, IcTKM and MODL has the following characteristics: HP Pavilion g6 Notebook PC with Intel (R) Core (TM) i5-3230M CPU @ 2.60GHz, 8 Gb of RAM, 4 cores, and installed with a dual boot featuring Ubuntu 04/18/4 LTS (for SSDL and MODL) and Windows 8 (for IcTCK, as running its code requires a Matlab license). The execution time indicated below are defined as follow: For SSDL, it corresponds to the difference between the times at the beginning and at the end of the learning step provided by SYS.TIME R function. For MODL, the execution time is defined similarly but results from the TIC/TOC python functions. Finally, IcTKM matlab code provides the execution duration in terms of CPU time.

4.2 Data description and data preprocessing

A full description of the Ecoli dataset as well as of its acquisition pipeline is available in [18]. An important feature of this pipeline is that the basic elements that are analyzed are peptides (*i.e.*, protein fragments). As described in Section 1, the matrix columns contain discrete chromatographic profiles obtained during the elution of the sample's peptides in liquid chromatography, while the rows represent mass spectra acquired at different

time stamps. The chemical properties of LCs are so that peptides with low masses are usually eluted at the beginning of the analysis, and heavy ones at the end. This leads to a specific matrix structure with high intensity peaks distributed along the diagonal, and with many zeros in the corners. This eases the data processing as it makes it possible to split the data matrix horizontally into several slightly overlapping slices, for which dictionaries can be independently trained. Then, since each slice contains a different set of peptides, the entire dictionary can be formed by concatenating the dictionaries from all the slices, which significantly reduces the computational cost.

The learning procedure being the same for all the slices, we report the experiments for a single one. We focus on a data slice of 718 rows acquired between 10 and 30 minutes (amongst the two hours that lasted the complete LC-MS analysis, *i.e.*, a quarter of the entire dataset). We have chosen this specific slice as for chemical reasons, it contains the highest density of MS peaks; making it the hardest part to extract a dictionary from. The resulting data matrix made of 74,193 columns (chromatographic profiles) has finally been sub-sampled uniformly at random from 718 to 256 rows.

4.3 Parameter tuning

SSDL method has eight parameters: the dictionary size K, the regularization parameter λ, the sketch size m, the batch size n, the initial learning rate γ_0, the decay parameter ν, the momentum weight α and the number of epochs T. In addition, SSDL requires an initial dictionary D^0, but it can easily be defined by a random selection from the data. Similarly, three epochs ($T = 3$) are practically sufficient to reach convergence on the Ecoli dataset.

Among the remaining seven parameters, a number of them should be tuned according to the specificities of the LC-MS pipeline and of the analyzed sample. Notably, the dictionary size must be consistent with the number of distinct peptides that are expected to be found. In our case, *E. Coli* being well studied, the number of peptides identified by a conventional mass spectrometry analysis is known to lie somewhere between 12,000 and 15,000, depending on the instrument and its tuning [18]. Thus, for a single slice broadly covering a quarter of the dataset, a suitable dictionary size can be estimated to lie between 3,000 and 3,750. However, to understand the effect of the dictionary size on the SSDL execution time, we also report smaller values of K. Notably, we hereafter discuss the influence of the batch size (see Figure 3a) in various scenarios with $K = \{384; 768; 1,536; 2,304; 3,072; 3,712\}$.

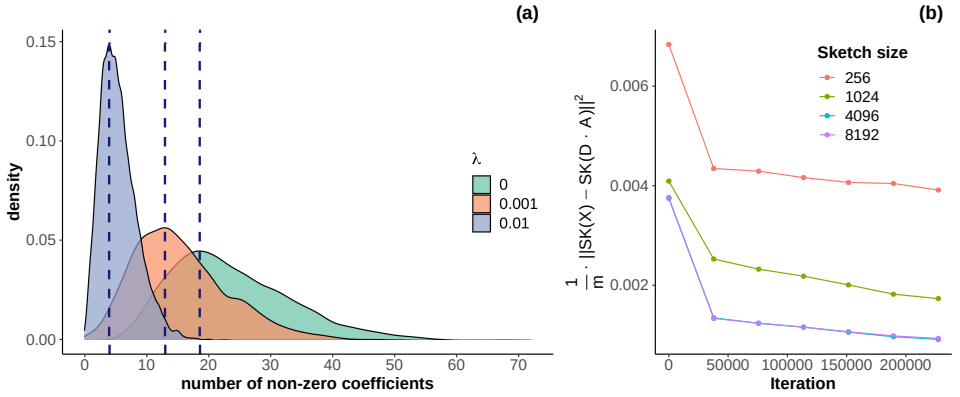

Figure 2: (a) Tuning of the regularization parameter λ. The distribution of the number of non-zero coefficients for different values of the regularization parameter λ. The coefficients are computed using the initial dictionary D^0. Vertical dotted lines indicate the mode of each distribution. (b) Value of the objective function of the dictionary update with respect to the number of iterations, with different sketch sizes.

The regularization parameter λ should be tuned so that the number of non-zero elements in the codes broadly amounts to the multiplexing level of the MS acquisitions. With these regards, it can be assumed [51] that on data such as those produced, chromatograms with more than 20 peaks are not sufficiently resoluted: they either correspond to noise, or to chemical species which disentanglement stands beyond the analytical power of the instrument. Thus, it is necessary to propose a practical strategy to find among various values of λ, the one that leads to the desired sparsity level. Concretely, we propose to consider a set of λ values on a logarithmized grid (*e.g.*, $\lambda \in \{0; 0.001; 0.01\}$), and for each of these values, to compute the code C^0 using the initial dictionary D^0. Figure 2a depicts the distribution of the number of non-zero elements in each column of C^0 for each λ: According to our expectations, $\lambda = 0.001$ is adapted to the Ecoli dataset. Although approximate, this approach appears to be sufficiently fast to be practically used by practitioners on their LC-MS dataset.

The tuning of the other five parameters is not as intuitive for a practitioner (sketch size, batch size, initial learning rate, decay parameter, and momentum weight). To mimic real conditions of application, it must therefore be carried out by means of a classical grid search. Let us focus on the sketch size m first: in general, increasing it allows revealing more details

about the data distribution, but at the cost of a longer execution time. To illustrate this trade-off, Figure 2b displays the objective function value in function of different values of sketch size ($m = \{256; 1,024; 4,096; 8,192\}$). We observe that at some point, increasing m no longer reduces the objective function value (the lines depicting $m = 4,096$ and $m = 8,192$ superimpose), so that the additional computational time does not worth it. Hereafter, m is fixed to 4,096.

Naturally, using larger batches (parameter n) leads to a better approximation of the decomposition sketch $SK(D \cdot C^*)$ in Eq. (9); yet, from a computational viewpoint the trade-off is not obvious: with larger batches, fewer iterations are required, but each iteration is more computationally demanding. As during preliminary experiments (not shown), we have considered various powers of 2 as batch sizes ($2^{10} = 1,024$; $2^{11} = 2,048$; $2^{12} = 4,096$; $2^{13} = 8,192$; $2^{14} = 16,384$; $2^{15} = 32,768$), with hardly any impact on SSDL convergence, we henceforth focus on the execution time. Figure 3a depicts SSDL execution time as a function of the dictionary size K for different batch size tests. Figure 3b illustrates SSDL execution time averaged across the different possible values of K. Overall, SSDL execution time mostly amounts to that of sparse coding, and thanks to our parallelized implementation (see Section 3.3), the dictionary update time hardly depends on n for $n \geq 4,096$.

The remaining three parameters (initial learning rate γ_0, decay parameter ν and momentum weight α) influence the convergence speed. Let us compare the 27 combinations resulting from the following tunings: $\gamma_0 = \{0.05; 0.1; 0.2\}$, $\nu = \{0; 0.01; 0.1\}$ and $\alpha = \{0; 0.5; 0.9\}$. The tests with $\nu = 0$ correspond to the case of constant learning rate. Three momentum weight values $\alpha = \{0; 0.5; 0.9\}$ represent three scenarios, respectively: (1) the momentum is not involved in the dictionary update (*i.e.*, the classical stochastic mini-batch method), (2) the gradient and the momentum have equivalent weights; and (3) the momentum has the majority impact (situations where $\alpha > 0.9$ should not be considered as too high a momentum can be detrimental to the optimality of the solution [52, 53]). The results are presented in Figure 4. At first glance and irrespective of the other parameters, the fastest convergence is given by $\alpha = 0.9$. Moreover, there is an important gap in the convergence rate of the classical stochastic mini-batch scheme and the momentum based Nesterov scheme which advocates for $\alpha = 0.9$. Furthermore, the higher the initial learning rate γ_0, the lower the final value of the objective function. However, setting the initial learning rate to 0.2 leads to too large fluctuations in the objective function, as illustrated on the rightmost part of Figure 4. In this case, a large decay parameter $\nu = 0.1$

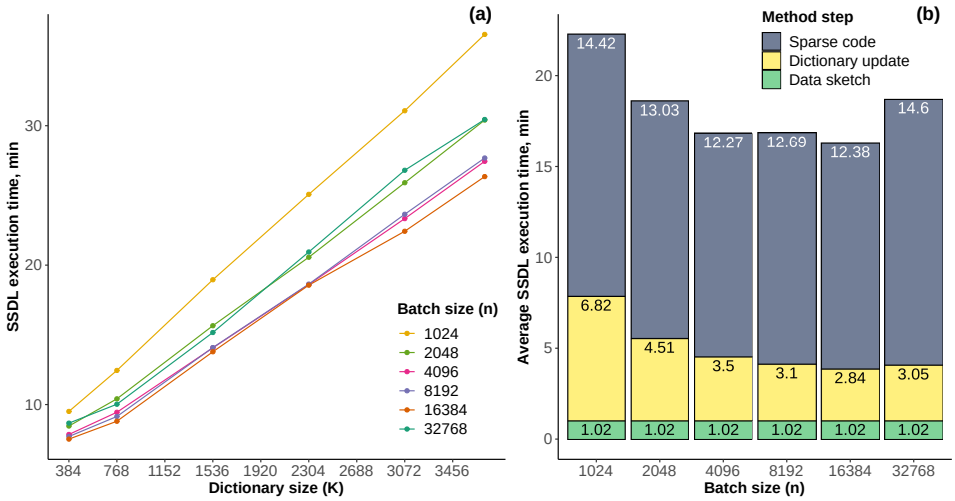

Figure 3: SSDL execution time for different batch size values. (a) The SSDL execution time as a function of the dictionary size and of the batch size. (b) Total average execution time as a function of the batch size as well as average execution time for each step among: data sketch computation (green); dictionary update (yellow); and sparse coding (blue).

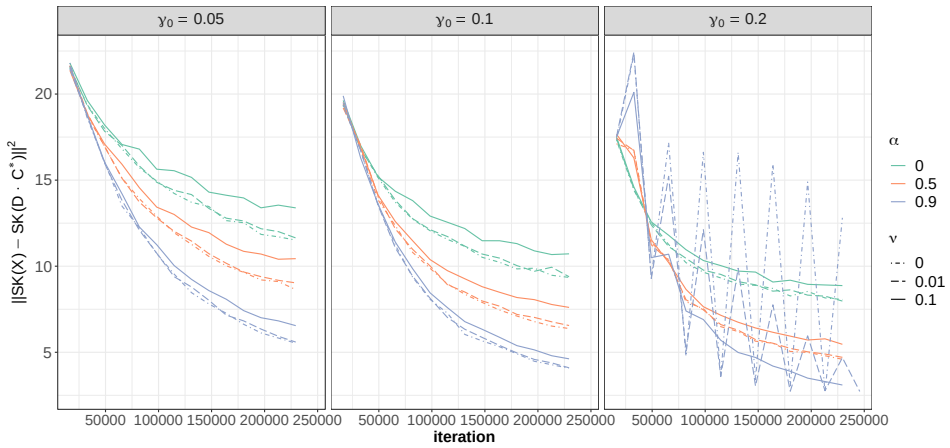

Figure 4: SSDL method convergence depending on the learning rate, the momentum weight, and the decay parameter. Sub-figures correspond to different initial learning rate tests (the smallest on the left and the biggest on the right). Each sub-figure depicts the dictionary update objective function $F(D, C^*)$ as a function of the number of iterations for different parameter value combinations. Different colors depict the three momentum weight scenarios, and different line types illustrate different values of the decay parameter. Batch size is fixed at 16,384. The regularization parameter λ is equal to 0.001.

can correct for this. We also observe that using a decaying learning rate when the initial learning rate γ_0 is smaller than 0.2 does not improve the convergence rate, and even slows it down sometimes. Finally, the following tuning is retained: $\alpha = 0.9$, $\gamma_0 = 0.2$ and $\nu = 0.1$.

MODL is driven by five parameters: the reduction parameter r, the batch size n_{MODL}, the dictionary size K, the regularization parameter λ_{MODL} and the number epochs T_{MODL}. The last three parameters being the same as for SSDL, they are set to the same values. As for the batch size, it is fixed to a value equal to that of the dictionary size K, as recommended in [33]. Finally, two scenarios are compared for the reduction level r: (1) the direct application of MODL to the original slice of 718 rows with $r = 3$ (referred to as $MODL_{718,r=3}$); and (2) the application of MODL after sub-sampling the slice down to 256 rows with $r = 1$ (referred to as $MODL_{256,r=1}$).

The parameters of IcTKM are selected following the recommendations of [35]: random projector based on discrete Fourier transform, sparsity level Λ_{IcTKM} set to the same value as expected for SSDL and reduction parameter $r_{IcTKM} = 5$ (the highest value according to [35], Table 1, based on the sparsity level and data dimension). As preliminary tests have highlighted the important computational load of IcTKM, we combine this dimensionality reduction with our subsampling to 256 rows, and the associated results are denoted as those of $IcTKM_{256,r=5}$. As for algorithm termination, instead of a number of epochs, IcTKM requires to fix the number of iterations. On our data, 32 of them seem sufficient to near the convergence plateau. Finally, concerning the initialization required for all the considered methods, dictionary D^0 is defined by a random selection from the data.

4.4 Results

Our comparisons focus on the execution time as well as on the quality of the resulting dictionaries with respect to the expectations listed in Section 1. Figure 5a depicts the execution time of $\text{MODL}_{256,r=1}$, $\text{MODL}_{718,r=3}$, $\text{IcTKM}_{256,r=5}$ and SSDL (which for the symmetry is denoted as SSDL_{256} with respect to the data preprocessing described in Section 4.2) depending on the dictionary size $K = \{384; 768; 1,152; 1,536; 1,920; 2,304; 2,688; 3,072; 3,712\}$, see Section 4.3. Other SSDL_{256} parameters are: $\lambda = 0.001$, $m = 4,096$, $n = 16,384$, $\gamma_0 = 0.2$, $\nu = 0.1$, $\alpha = 0.9$, $T = 3$.

Despite the combination of both dimensionality reduction methods, $\text{IcTKM}_{256,r=5}$ is the slowest approach. MODL is more computationally efficient than SSDL for small dictionaries (broadly, less than 1,000-1,500 atoms for $\text{MODL}_{256,r=1}$ and less than 768 for $\text{MODL}_{718,r=3}$), but it does not easily scale up to too

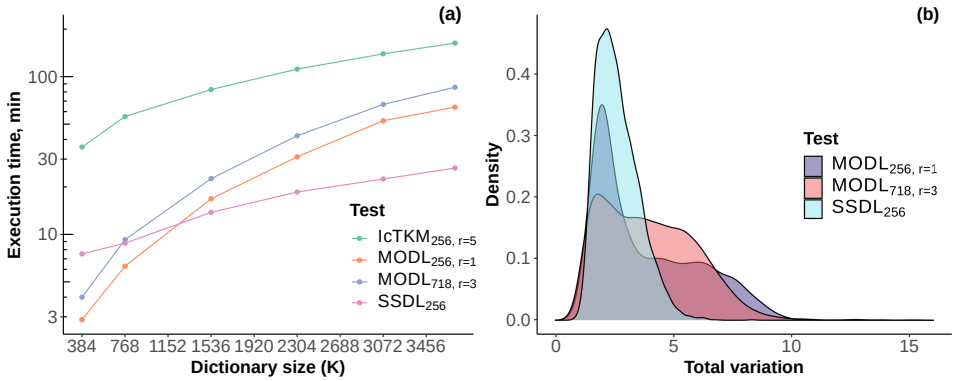

Figure 5: (a) The execution time (logarithmic scale) of SSDL, MODL, IcTKM tests as a function of the dictionary size K. The execution time of all methods consists in the execution time spent on the sparse coding and the dictionary update. However, for SSDL (resp. IcTKM) it also includes the data sketch computation time (resp. the random projection operator construction time). (b) The distribution of the dictionary atom total variation for $\text{MODL}_{256,r=1}$ (purple), $\text{MODL}_{718,r=3}$ (red) and SSDL_{256} (light blue) resulting dictionaries (with $K = 3,712$).

large dictionaries. As a result, for datasets resulting from the LC-MS analysis of highly complex biological samples, SSDL is more efficient: Concretely, for a single slice of Ecoli dataset, for which $K \in [3000, 3750]$), SSDL_{256} test is two times faster than $\text{MODL}_{256,r=1}$, three times than $\text{MODL}_{718,r=3}$ and four times than $\text{IcTKM}_{256,r=5}$.

Concerning the dictionary quality, as discussed in Section 1, the dictionary atoms must have shapes akin to that of real chromatographic profiles: positive and smooth signals with a Gaussian like, yet slightly asymmetric, shape. Figure 6 illustrates with several examples, the type of dictionary atoms obtained by SSDL_{256} (first row), $\text{MODL}_{256,r=1}$ (second row) and $\text{IcTKM}_{256,r=5}$ (third row). Since IcTKM method does not allow to impose any positiveness or smoothness constraints, the obtained dictionary atoms cannot be interpreted as peptide chromatograms, hereby hampering their use for processing multiplexed acquisitions. In contrast, both SSDL and MODL provide dictionary atoms with the expected chromatogram shape. Of course, both SSDL and MODL also provides atoms that cannot be interpreted as single chromatogram: for instance, the fourth dictionary atom in the first row of Figure 6 contains many well-separated peaks; and the

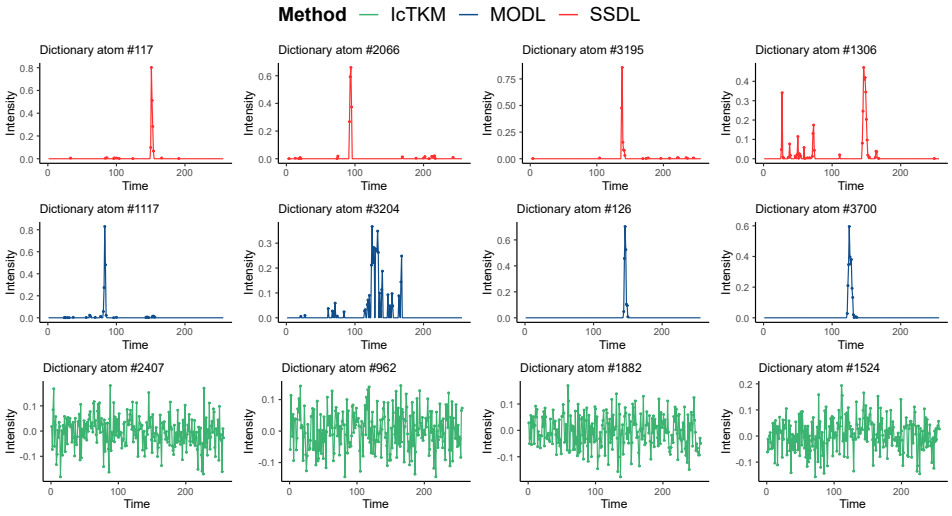

Figure 6: The examples of the resulting dictionary atoms (with $K = 3,712$) obtained by SSDL_{256} (first row), $\text{MODL}_{256,r=1}$ (second row) and $\text{IcTKM}_{256,r=5}$ (third row).

second one in the second row of Figure 6 displays what appears to be an overlap of various chromatograms. However, this is simply a consequence of the biological complexity of the analyzed sample, which may require more resoluted instruments as well as, possibly, further improvements in blind source separation.

To obtain a more precise and more exhaustive comparison of MODL and SSDL dictionaries, displaying the total variation distributions is insightful: Since the dictionary atoms have a unitary L^2 norm, chromatogram-like atoms should have a total variation smaller than 2. Considering the total variation of the j^{th} atom of dictionary D reads

$$TV(j) = \sum_{k=1}^{s-1} |D_{k+1,j} - D_{k,j}|, \tag{14}$$

we can compare the histograms of the three $TV(j)_{j\in[1,K]}$ distributions derived from learning D with $\text{MODL}_{256,r=1}$, $\text{MODL}_{718,r=3}$ and SSDL_{256}. The result is depicted on Figure 5b, with density plots practically obtained using the `geom_density` function of ggplot2 R package (default parameters). It appears that the dictionaries extracted with both $\text{MODL}_{256,r=1}$ and $\text{MODL}_{718,r=3}$ contain many atoms with a total variation norm around

2. However the distributions are also heavy-tailed, with a significant proportion of less smooth atoms (total variation norm lying between 3 and 10). In contrast, the total variation norm distribution for SSDL_{256} does not exceed 6, and indicates that a larger proportion of the atoms are smoother than their MODL counterparts. Overall, SSDL provides with a smaller computational time, dictionaries that are more suited to LC-MS data than those produced with MODL and IcTKM.

5 Conclusions

Extracting meaningful patterns from LC-MS data is challenging, because of the multiple constraints attached to their production method: First, the corresponding matrix can be of very large size, especially when resulting from last generation high resolution instruments, hereby requiring scalable approaches. Second, the extracted patterns must have the physical interpretation of a chromatogram and their number must be consistent with the number of analytes potentially detected in the analyzed sample (up to several thousands). In this article, we have introduced a new dictionary learning method, referred to as Sketched Stochastic Dictionary Learning (SSDL), which combines the latest trends of compressive statistical learning, as well as of online learning and of stochastic optimization, while being compliant with all the aforementioned constraints. This is notably the reason why, compared to state-of-the-art methods, it provides more meaningful dictionaries at a smaller computational cost. Beyond the specificities of LC-MS data, SSDL is also of interest from a more fundamental viewpoint, as to the best of our knowledge, it is the first dictionary learning method that can directly operate on a data sketch (a controlled-sized proxy of the data distribution in the Fourier domain). As future work, we will consider embedding random projection based dimensionality reduction techniques, hereby enabling the processing of entire datasets in a single batch; as well as eventually, the processing of datasets acquired on longer time frames with longer elution columns. From a more applicative viewpoint, SSDL will unleash an efficient and convenient handling of highly multiplexed data acquisitions. Such acquisitions are already an important research direction in proteomics for the depth of analysis they potentially enable, however to date the associated data processing challenges have prevented their widespread use; a hurdle that SSDL will help to overcome.

Acknowledgments

The authors would like to thank Anne-Marie Hesse and Alexandra Kraut for carrying out the mass spectrometry experiments that provided the data on which this paper is based, as well as Virginie Brun, Yohann Couté and Christophe Bruley for supports and fruitful discussions.

Data Accessibility

The data that support the findings of this study are openly available in figshare public repository at http://doi.org/10.6084/m9.figshare.13589621.

Author contributions

Olga Permiakova designed the method, implemented the R package, carried out the computational experiments, analysed the results and drafted the manuscript. Thomas Burger designed the method, directed the work, participated to the result analysis and drafted the manuscript. All authors proofread the manuscript and approved its final version.

Financial disclosure

This work was supported by grants from the French National Research Agency: ProFI project (ANR-10- INBS-08), GRAL project (ANR-10-LABX-49-01), DATA@UGA and SYMER projects (ANR-15-IDEX-02) and MIAI @ Grenoble Alpes (ANR-19-P3IA-0003).

Conflict of interest

The authors declare no potential conflict of interests.

References

[1] Christian Jutten and Jeanny Herault. Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture. *Signal processing*, 24(1):1–10, 1991.

[2] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[3] Michael Elad. *Sparse and redundant representations: from theory to applications in signal and image processing*. Springer Science & Business Media, 2010.

[4] Shutao Li, Haitao Yin, and Leyuan Fang. Group-sparse representation with dictionary learning for medical image denoising and fusion. *IEEE Transactions on biomedical engineering*, 59(12):3450–3459, 2012.

[5] Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. Image super-resolution via sparse representation. *IEEE transactions on image processing*, 19(11):2861–2873, 2010.

[6] Michael Lustig, David Donoho, and John M Pauly. Sparse mri: The application of compressed sensing for rapid mr imaging. *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, 58(6):1182–1195, 2007.

[7] S. G. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, 1989.

[8] David L Donoho, Michael Elad, and Vladimir N Temlyakov. Stable recovery of sparse overcomplete representations in the presence of noise. *IEEE Transactions on information theory*, 52(1):6–18, 2005.

[9] Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[10] Jinglei Lv, Xi Jiang, Xiang Li, Dajiang Zhu, Hanbo Chen, Tuo Zhang, Shu Zhang, Xintao Hu, Junwei Han, Heng Huang, et al. Sparse representation of whole-brain fmri signals for identification of functional networks. *Medical image analysis*, 20(1):112–134, 2015.

[11] Patrik O Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5(Nov):1457–1469, 2004.

[12] M. Elad and M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing*, 15(12):3736–3745, 2006.

[13] Mark D Plumbley, Thomas Blumensath, Laurent Daudet, Rémi Gribonval, and Mike E Davies. Sparse representations in audio and music: from coding to source separation. *Proceedings of the IEEE*, 98(6):995–1005, 2009.

[14] Thomas E Angel, Uma K Aryal, Shawna M Hengel, Erin S Baker, Ryan T Kelly, Errol W Robinson, and Richard D Smith. Mass spectrometry-based proteomics: existing capabilities and future directions. *Chemical Society Reviews*, 41(10):3912–3928, 2012.

[15] John D Chapman, David R Goodlett, and Christophe D Masselon. Multiplexed and data-independent tandem mass spectrometry for global proteome profiling. *Mass spectrometry reviews*, 33(6):452–470, 2014.

[16] Ryan Peckner, Samuel A Myers, Alvaro Sebastian Vaca Jacome, Jarrett D Egertson, Jennifer G Abelin, Michael J MacCoss, Steven A Carr, and Jacob D Jaffe. Specter: linear deconvolution for targeted analysis of data-independent acquisition mass spectrometry proteomics. *Nature methods*, 15(5):371, 2018.

[17] Jérémy Rapin, Antoine Souloumiac, Jérôme Bobin, Anthony Larue, Chistophe Junot, Minale Ouethrani, and Jean-Luc Starck. Application of non-negative matrix factorization to lc/ms data. *Signal Processing*, 123:75–83, 2016.

[18] Olga Permiakova, Romain Guibert, Alexandra Kraut, Thomas Fortin, Anne-Marie Hesse, and Thomas Burger. Chickn: Extraction of peptide chromatographic elution profiles from large scale mass spectrometry data by means of wasserstein compressive hierarchical cluster analysis. *BMC Bioinformatics (under revision)*, 2020.

[19] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[20] David L Donoho and Michael Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via l1 minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202, 2003.

[21] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[22] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[23] Dar Gilboa, Sam Buchanan, and John Wright. Efficient dictionary learning with gradient descent. In *International Conference on Machine Learning*, pages 2252–2259, 2019.

[24] Bao-Di Liu, Yu-Xiong Wang, Bin Shen, Xue Li, Yu-Jin Zhang, and Yan-Jiang Wang. Blockwise coordinate descent schemes for efficient and effective dictionary learning. *Neurocomputing*, 178:25–35, 2016.

[25] Kjersti Engan, Sven Ole Aase, and J Hakon Husoy. Method of optimal directions for frame design. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, volume 5, pages 2443–2446. IEEE, 1999.

[26] Michal Aharon, Michael Elad, and Alfred Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311–4322, 2006.

[27] Igor Fedorov, Bhaskar D Rao, and Truong Q Nguyen. Multimodal sparse bayesian dictionary learning applied to multimodal data classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2237–2241. IEEE, 2017.

[28] Geethu Joseph and Chandra R Murthy. On the convergence of a bayesian algorithm for joint dictionary learning and sparse recovery. *IEEE Transactions on Signal Processing*, 68:343–358, 2019.

[29] Linxiao Yang, Jun Fang, Hong Cheng, and Hongbin Li. Sparse bayesian dictionary learning with a gaussian hierarchical model. *Signal Processing*, 130:93–104, 2017.

[30] Morgan A Schmitz, Matthieu Heitz, Nicolas Bonneel, Fred Ngole, David Coeurjolly, Marco Cuturi, Gabriel Peyré, and Jean-Luc Starck. Wasserstein dictionary learning: Optimal transport-based unsupervised nonlinear dictionary learning. *SIAM Journal on Imaging Sciences*, 11(1):643–678, 2018.

[31] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[32] Konstantinos Slavakis and Georgios B Giannakis. Online dictionary learning from big data using accelerated stochastic approximation algorithms. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 16–20. IEEE, 2014.

[33] Arthur Mensch, Julien Mairal, Bertrand Thirion, and Gaël Varoquaux. Stochastic subsampling for factorizing huge matrices. *IEEE Transactions on Signal Processing*, 66(1):113–128, 2017.

[34] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(Jan):19–60, 2010.

[35] Karin Schnass and Flavio Teixeira. Compressed dictionary learning. *Journal of Fourier Analysis and Applications*, 26(2):1–37, 2020.

[36] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250, 2001.

[37] Thomas Blumensath and Mike E Davies. Iterative thresholding for sparse approximations. *Journal of Fourier analysis and Applications*, 14(5-6):629–654, 2008.

[38] Chris Ding, Xiaofeng He, and Horst D Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 606–610. SIAM, 2005.

[39] Nir Ailon and Bernard Chazelle. The fast johnson–lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on computing*, 39(1):302–322, 2009.

[40] Nicolas Keriven, Anthony Bourrier, Rémi Gribonval, and Patrick Pérez. Sketching for large-scale learning of mixture models. *Information and Inference: A Journal of the IMA*, 7(3):447–508, 2018.

[41] Nicolas Keriven, Nicolas Tremblay, Yann Traonmilin, and Rémi Gribonval. Compressive k-means. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6369–6373. IEEE, 2017.

[42] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[43] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[44] Mahmoud Assran and Michael Rabbat. On the convergence of nesterov's accelerated gradient method in stochastic settings. *arXiv preprint arXiv:2002.12414*, 2020.

[45] Andrei Kulunchakov and Julien Mairal. A generic acceleration framework for stochastic composite optimization. In *Advances in Neural Information Processing Systems*, pages 12556–12567, 2019.

[46] Dirk Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer, New York, 2013. ISBN 978-1-4614-6867-7.

[47] JJ Allaire, R Francois, K Ushey, G Vandenbrouck, and M Geelnard. Rcppparallel: Parallel programming tools for "rcpp". *R package version*, 4:20, 2016.

[48] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

[49] Markus Schmidberger, Martin Morgan, Dirk Eddelbuettel, Hao Yu, Luke Tierney, and Ulrich Mansmann. State-of-the-art in parallel computing with r. *Journal of Statistical Software*, 47(1), 2009.

[50] Florian Privé, Hugues Aschard, Andrey Ziyatdinov, and Michael G.B. Blum. Efficient analysis of large-scale genome-wide data with two R packages: bigstatsr and bigsnpr. *Bioinformatics*, 34(16):2781–2787, 2018.

[51] Ingvar Eidhammer, Kristian Flikka, Lennart Martens, and Svein-Ole Mikalsen. *Computational methods for mass spectrometry proteomics*. Wiley Online Library, 2007.

[52] Leslie N. Smith. A disciplined approach to neural network hyperparameters: Part 1 – learning rate, batch size, momentum, and weight decay, 2018.

[53] Tianyi Liu, Zhehui Chen, Enlu Zhou, and Tuo Zhao. A diffusion approximation theory of momentum sgd in nonconvex optimization, 2021.