MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Sparse NMF – half-baked or well done?

Le Roux, J.; Weninger, F.J.; Hershey, J.R.

TR2015-023 March 2015

Abstract

Non-negative matrix factorization (NMF) has been a popular method for modeling audio signals, in particular for single-channel source separation. An important factor in the success of NMF-based algorithms is the "quality" of the basis functions that are obtained from training data. In order to model rich signals such as speech or wide ranges of non-stationary noises, NMF typically requires using a large number of basis functions. However, without additional constraints, using a large number of bases leads to trivial solutions where the bases can indiscriminately model any signal. Two main approaches have been considered to cope with this issue: introducing sparsity on the activation coefficients, or skipping training altogether and randomly selecting basis functions as a subset of the training data ("exemplarbased NMF"). Surprisingly, the sparsity route is widely regarded as leading to similar or worse results than the simple and extremely efficient (no training!) exemplar-based approach. Only a small fraction of researchers have realized that sparse NMF works well if implemented correctly. However, to our knowledge, no thorough comparison has been presented in the literature, and many researchers in the field may remain unaware of this fact. We review exemplar-based NMF as well as two versions of sparse NMF, a simplistic ad hoc one and a principled one, giving a detailed derivation of the update equations for the latter in the general case of beta divergences, and we perform a thorough comparison of the three methods on a speech separation task using the 2nd CHiME Speech Separation and Recognition Challenge dataset. Results show that, contrary to a popular belief in the community, learning basis functions using NMF with sparsity, if done the right way, leads to significant gains in source-to-distortion ratio with respect to both exemplar-based NMF and the ad hoc implementation of sparse NMF.

Mitsubishi Electric Research Laboratories

Sparse NMF – half-baked or well done?

Jonathan Le Roux
Mitsubishi Electric Research Labs (MERL)
Cambridge, MA, USA
leroux@merl.com

Felix Weninger
TUM
Munich, Germany
felix@weninger.de

John R. Hershey
Mitsubishi Electric Research Labs (MERL)
Cambridge, MA, USA
hershey@merl.com

Abstract

Non-negative matrix factorization (NMF) has been a popular method for modeling audio signals, in particular for single-channel source separation. An important factor in the success of NMF-based algorithms is the "quality" of the basis functions that are obtained from training data. In order to model rich signals such as speech or wide ranges of non-stationary noises, NMF typically requires using a large number of basis functions. However, without additional constraints, using a large number of bases leads to trivial solutions where the bases can indiscriminately model any signal. Two main approaches have been considered to cope with this issue: introducing sparsity on the activation coefficients, or skipping training altogether and randomly selecting basis functions as a subset of the training data ("exemplar-based NMF"). Surprisingly, the sparsity route is widely regarded as leading to similar or worse results than the simple and extremely efficient (no training!) exemplar-based approach. Only a small fraction of researchers have realized that sparse NMF works well if implemented correctly. However, to our knowledge, no thorough comparison has been presented in the literature, and many researchers in the field may remain unaware of this fact. We review exemplar-based NMF as well as two versions of sparse NMF, a simplistic ad hoc one and a principled one, giving a detailed derivation of the update equations for the latter in the general case of beta divergences, and we perform a thorough comparison of the three methods on a speech separation task using the 2nd CHiME Speech Separation and Recognition Challenge dataset. Results show that, contrary to a popular belief in the community, learning basis functions using NMF with sparsity, if done the right way, leads to significant gains in source-to-distortion ratio with respect to both exemplar-based NMF and the ad hoc implementation of sparse NMF.

1 Contributions of this report

- Experimental comparison of exemplar-based NMF (ENMF), sparse NMF with basis renormalization in the objective function (SNMF), sparse NMF with basis renormalization after each update (NMF+S) on a supervised audio source separation task: **new**
- Detailed derivation of multiplicative update equations for SNMF with beta divergence from a general perspective of gradients with unit-norm constraints: **discussion on gradients with unit-norm constraints adapted from our previous work [1]; some elements similar to the derivation for the convolutive NMF case in [2]**

- Proof that the tangent gradient and the natural gradient are equivalent for the unit L^2-norm constraint: **new**
- An efficient MATLAB® implementation of SNMF with β-divergence: **new**

2 Non-negative matrix factorization and sparsity

Non-negative matrix factorization (NMF) is a popular approach to analyzing non-negative data. It has been successfully used over the past 15 years in a surprisingly wide range of applications (e.g., [3, 4] and references therein). In many cases, it is used as a dimension reduction technique, where the goal is to factor some non-negative data matrix $\mathbf{M} \in \mathbb{R}^{F \times T}$ into the product of two non-negative matrices $\mathbf{W} \in \mathbb{R}^{F \times R}$ and $\mathbf{H} \in \mathbb{R}^{R \times T}$, where the inner dimension R of the product is much smaller than either dimensions F and T of the original matrix. Without any sparsity penalty on either or both of the factors $\mathbf{W}$ and $\mathbf{H}$, applying NMF with R of the order of, or larger than F or T would lead to meaningless results, as there are infinitely many different factorizations that can exactly lead to $\mathbf{M}$. In the context of audio signal processing for example, NMF is typically applied to the magnitude or power spectrogram $\mathbf{M}$ of a corpus of sounds. The matrix $\mathbf{W}$ can then be interpreted as a set of spectral basis functions (also referred to as a dictionary), and the matrix $\mathbf{H}$ as the collection of their activations at each time frame. However, whatever the size of the corpus, vanilla NMF will be unable to learn a meaningful dictionary with more elements than there are time frequency bins. To alleviate this issue, one can introduce sparsity penalties on the factors, for example on the activations $\mathbf{H}$ in the context of audio. This ensures that, even though there are many elements in the dictionary, only a small number will be active at the same time. This enables the model to learn a meaningful representation of the data with many more elements than what vanilla NMF would allow. Another approach to solving this issue of representing rich datasets is to obtain the dictionary not by learning, but by sampling the data, i.e., by using randomly selected data samples as the dictionary elements. Both approaches have been explored. We present here an overview of these approaches, and thoroughly compare their performance on a supervised audio source separation task.

In unsupervised application scenarios, both $\mathbf{W}$ and $\mathbf{H}$ are optimized at test time so that their product fits the data; in supervised application scenarios, $\mathbf{W}$ is typically learned at training time, again by optimizing $\mathbf{W}$ and $\mathbf{H}^{\text{training}}$ so that their product fits some training data [2, 5], while at test time, only $\mathbf{H}$ is optimized given $\mathbf{W}$. In any case, both types of scenario involve solving the following problem:

$$\overline{\mathbf{W}}, \overline{\mathbf{H}} = \arg\min_{\mathbf{W},\mathbf{H}} D(\mathbf{M} \mid \mathbf{WH}) + \mu|\mathbf{H}|_1, \tag{1}$$

where D is a cost function that is minimized when $\mathbf{M} = \mathbf{WH}$. We abuse notations on D and assume that the cost function computed on matrices is equal to the sum of the element-wise costs over all elements. Here we use the β-divergence, D_β, which for $\beta = 0$ yields the Itakura-Saito (IS) distance, for $\beta = 1$ yields the generalized Kullback-Leibler (KL) divergence, and for $\beta = 2$ yields the Euclidean distance. The β-divergence is defined in general as:

$$D_\beta(x|y) \stackrel{\text{def}}{=} \begin{cases} \frac{1}{\beta(\beta-1)}\left(x^\beta - y^\beta - \beta y^{\beta-1}(x-y)\right) & \text{if } \beta \in \mathbb{R}\backslash\{0,1\} \\ x(\log x - \log y) + (y - x) & \text{if } \beta = 1 \\ \frac{x}{y} - \log\frac{x}{y} - 1 & \text{if } \beta = 0 \end{cases}. \tag{2}$$

The L^1 sparsity constraint with weight μ is added to favor solutions where few basis vectors are active at a time. To avoid scaling indeterminacies, some normalization constraint is typically assumed on $\mathbf{W}$, e.g., $||W||_2 = 1$ where $||\cdot||_2$ denotes the L^2 norm.

If the NMF bases $\mathbf{W}$ are held fixed, such as at test time in supervised scenarios, the optimal activations $\hat{\mathbf{H}}$ are estimated such that

$$\hat{\mathbf{H}} = \arg\min_{\mathbf{H}} D(\mathbf{M} \mid \mathbf{WH}) + \mu|\mathbf{H}|_1. \tag{3}$$

A convenient algorithm [6] for minimizing (3) that preserves non-negativity of $\mathbf{H}$ by multiplicative updates is given by iterating

$$\mathbf{H} \leftarrow \mathbf{H} \otimes \frac{\mathbf{W}^\intercal(\mathbf{M} \otimes \mathbf{\Lambda}^{\beta-2})}{\mathbf{W}^\intercal\mathbf{\Lambda}^{\beta-1} + \mu} \tag{4}$$

until convergence, with $\mathbf{\Lambda} := \mathbf{WH}$ and exponents applied element-wise.

It remains to determine how to obtain the NMF bases $\mathbf{W}$. In NMF without a sparsity cost on $\mathbf{H}$, i.e. $\mu = 0$, an optimization scheme that alternates between updating $\mathbf{H}$ given $\mathbf{W}$ as in (4):

$$\mathbf{H} \leftarrow \mathbf{H} \otimes \frac{\mathbf{W}^\intercal\big(\mathbf{M} \otimes \mathbf{\Lambda}^{\beta-2}\big)}{\mathbf{W}^\intercal \mathbf{\Lambda}^{\beta-1}}, \tag{5}$$

and updating $\mathbf{W}$ given $\mathbf{H}$ in a similar way:

$$\mathbf{W} \leftarrow \mathbf{W} \otimes \frac{\big(\mathbf{\Lambda}^{\beta-2} \otimes \mathbf{M}\big)\mathbf{H}^\intercal}{\mathbf{\Lambda}^{\beta-1}\mathbf{H}^\intercal}, \tag{6}$$

where $\mathbf{\Lambda}$ is kept updated to $\mathbf{WH}$, can be shown to be non-decreasing with respect to the objective function [6]. However, optimizing (1) including the sparsity cost requires special care concerning the normalization constraint on $\mathbf{W}$, as we shall see below. Another option that has been explored is to avoid training altogether and simply select the NMF bases $\mathbf{W}$ randomly from training samples. We shall now investigate these options in more details.

3 Obtaining NMF Bases

3.1 Learning the bases: Sparse NMF (SNMF) or NMF with sparsity (NMF+S)

Since the L^1 sparsity constraint on $\mathbf{H}$ in (1) is not scale-invariant, it can be trivially minimized by scaling of the factors. As mentioned above, this scaling indeterminacy can be avoided by imposing some sort of normalization on the bases $\mathbf{W}$. Two approaches for enforcing this normalization have been predominant in the literature.

A tempting but ad hoc practice [7] is to perform standard NMF optimization with sparsity in the $\mathbf{H}$ update, using (4) then (6), and rescaling $\mathbf{W}$ and $\mathbf{H}$ such that $\mathbf{W}$ has unit norm after each iteration. We here denote this approach NMF+S, for NMF with sparsity.

Another approach [2, 8] is to directly reformulate the objective function including a column-wise normalized version of $\mathbf{W}$ [9], leading to an approach which we refer to as sparse NMF (SNMF):

$$\overline{\mathbf{W}}, \overline{\mathbf{H}} = \underset{\mathbf{W},\mathbf{H}}{\arg\min}\, D_\beta(\mathbf{S} \mid \widetilde{\mathbf{W}}\mathbf{H}) + \mu|\mathbf{H}|_1, \tag{7}$$

where $\widetilde{\mathbf{W}} = \left[\frac{\mathbf{w}_1}{\|\mathbf{w}_1\|} \cdots \frac{\mathbf{w}_R}{\|\mathbf{w}_R\|}\right]$ is the column-wise normalized version of $\mathbf{W}$. The update for $\mathbf{H}$ given $\mathbf{W}$ is essentially the same as before, except that it now involves $\widetilde{\mathbf{W}}$ instead of $\mathbf{W}$:

$$\mathbf{H} \leftarrow \mathbf{H} \otimes \frac{\widetilde{\mathbf{W}}^\intercal(\mathbf{M} \otimes \mathbf{\Lambda}^{\beta-2})}{\widetilde{\mathbf{W}}^\intercal \mathbf{\Lambda}^{\beta-1} + \mu},$$

now with $\mathbf{\Lambda} := \widetilde{\mathbf{W}}\mathbf{H}$. However, the update for $\mathbf{W}$ given $\mathbf{H}$ will be different from (6), which was obtained without normalization. We derive this update equation in Section 5.

Importantly, as pointed out by [9], SNMF is not equivalent to NMF+S. Indeed, the a posteriori rescaling of $\mathbf{W}$ and $\mathbf{H}$ in NMF+S changes the value of the sparsity cost function, which is not the case for (7) as $\mathbf{H}$ is not rescaled. One can in fact see that the NMF+S optimization scheme may actually lead to an increase in the cost function, as shown in Figure 1. In the course of optimization with SNMF, the evolution of the KL divergence part and the sparsity part of the objective function shows that a trade-off between the two is reached, while the total cost function decreases monotonously; with NMF+S, an increase of the sparsity part and of the total cost function can be observed after a few iterations. In practice, the iterations are stopped when the total objective function starts increasing.

Multiplicative update algorithms to optimize (7) with the Euclidean distance and KL divergence have appeared in the literature. We only found the case of an arbitrary $\beta \geq 0$ presented in [2] for the case of convolutive NMF. We give here a detailed derivation for the case of NMF, using a similar perspective to [2], but including a more general account on optimization of objective function with normalized parameters.

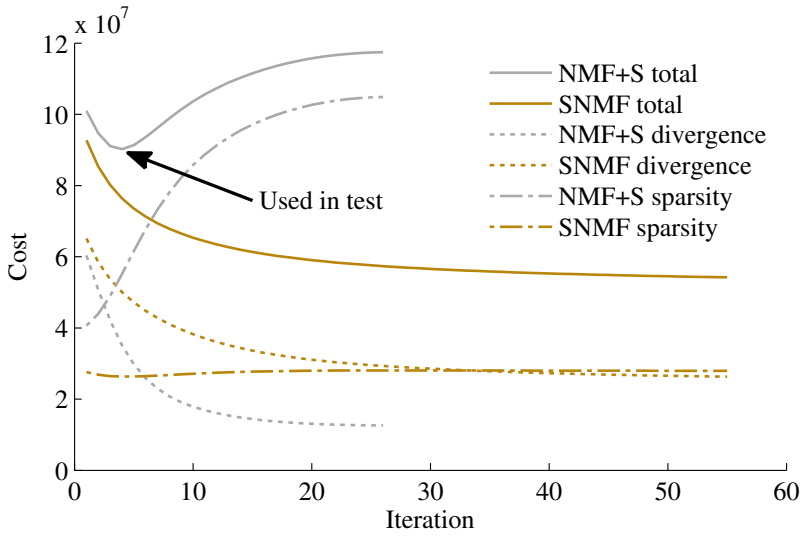


Figure 1: Evolution of the cost function for SNMF and NMF+S with the KL divergence ($\beta = 1$), for $\mu = 5$ and $K = 1000$, while training on a database of speech power spectra as described in Section 7.

3.2 Sampling the bases: Exemplar-based NMF (ENMF)

Recently, exemplar-based approaches, where every basis function corresponds to an observation in the training data, have become popular for large-scale factorizations of audio signals [10, 11]. It has been argued that they lead to similar or better performance compared with NMF with sparsity, with the extra advantage that they do not require training [12] (see also [13] for a similar study on complex NMF [14]). It is however hard to find thorough comparisons of ENMF with NMF methods where the basis functions are trained using sparsity constraints. We will show experimentally on a large supervised audio source separation task that, even though it may have been true that ENMF performed better when compared with the ad-hoc NMF+S, it is actually not the case anymore if compared with SNMF, the proper implementation of sparsity in NMF.

4 Optimization of objective functions with normalized parameters

We explained above how in NMF+S the update of $\mathbf{W}$ is performed in two steps: first a multiplicative update without normalization constraint, followed by normalization to ensure that the bases $\mathbf{W}$ respect the unit-norm constraint. A big issue with this approach is that it requires a corresponding rescaling of $\mathbf{H}$, and this rescaling may increase the contribution of the sparsity term to the objective function, potentially resulting in an increase of the objective function compared to the value before normalization of $\mathbf{W}$. Even without sparsity term, while the normalization of $\mathbf{W}$ and rescaling of $\mathbf{H}$ do not change the value of the objective function, alleviating the scaling indeterminacy between $\mathbf{W}$ and $\mathbf{H}$ may still help the optimization process. Updating $\mathbf{W}$ while taking into account as much as possible the unit-norm normalization on the bases is thus important with or without a sparsity penalty.

A heuristic approach to deriving multiplicative update equations for NMF relies on computing the gradient of the objective function and splitting it into the difference of two non-negative terms. We thus consider here the general problem of computing a notion of the gradient of an objective function with normalized parameters. Two notions of gradient appear to be natural candidates for consideration here: the natural gradient [15] of the objective function with (non-normalized) parameters on the unit-norm constraint manifold, and the classical gradient (w.r.t. the non-normalized parameters) of the objective function in which the bases appear explicitly normalized [9, 16]. It turns out that for a unit-norm normalization with the L^2-norm, these two notions of gradient are actually equivalent.

4.1 Gradient with normalized parameters

In general, let $\mathcal{I}(w)$ be an objective function with parameter $w \in \mathbb{R}^n$, and let $\widetilde{\mathcal{I}}(w) = \mathcal{I}(\frac{w}{||w||})$, where $||w||$ is any vector norm, be the corresponding objective function where the parameters are explicitly normalized. In the following, we shall denote $v = \frac{\partial ||u||}{\partial u}\Big|_{u=w}$ and $\widetilde{w} = \frac{1}{||w||}w$. Then

$$\frac{\partial \widetilde{\mathcal{I}}}{\partial w_i}(w) = \sum_j \frac{\partial \widetilde{w}_j}{\partial w_i} \cdot \frac{\partial \mathcal{I}}{\partial w_j}(\widetilde{w}), \tag{8}$$

and using

$$\frac{\partial \widetilde{w}_j}{\partial w_i} = \begin{cases} \frac{1}{||w||} - \frac{w_i v_i}{||w||^2} & \text{if } j = i \\ -\frac{w_j v_i}{||w||^2} & \text{if } j \neq i \end{cases} \tag{9}$$

we obtain

$$\frac{\partial \widetilde{\mathcal{I}}}{\partial w_i}(w) = \frac{1}{||w||}\frac{\partial \mathcal{I}}{\partial w_i}(\widetilde{w}) - v_i \sum_j \frac{w_j}{||w||^2} \cdot \frac{\partial \mathcal{I}}{\partial w_j}(\widetilde{w}), \tag{10}$$

and finally

$$\nabla \widetilde{\mathcal{I}}(w) = \frac{1}{||w||}\Big(\nabla \mathcal{I}(\widetilde{w}) - (\widetilde{w} \cdot \nabla \mathcal{I}(\widetilde{w}))v\Big), \tag{11}$$

or written in matrix form

$$\nabla \widetilde{\mathcal{I}}(w) = \frac{1}{||w||}\left(\mathrm{Id} - v\widetilde{w}^\intercal\right)\nabla \mathcal{I}(\widetilde{w}). \tag{12}$$

For the L^2 norm, we have $v = \frac{w}{||w||}$, and Eq. (12) simplifies to

$$\nabla \widetilde{\mathcal{I}}(w) = \frac{1}{||w||}\left(\mathrm{Id} - vv^\intercal\right)\nabla \mathcal{I}(\widetilde{w}). \tag{13}$$

This is all we need to compute the update equations in Section 5, but we study further the relationship between this notion of gradient and the so-called tangent and natural gradients.

4.2 Relation to the tangent gradient

The tangent gradient, introduced formally in [17] (it had been used earlier, e.g., by Krasulina [18] or Reddy et al. [19]), is defined in general for a unit-norm constraint (with arbitrary norm) as the orthogonal projection of the gradient onto the tangent plane to the unit-norm surface at $\widetilde{w}$. It can be computed as

$$\nabla^{(T)}\mathcal{I}(\widetilde{w}) = \left(\mathrm{Id} - \frac{vv^\intercal}{||v||_2^2}\right)\nabla \mathcal{I}(\widetilde{w}), \tag{14}$$

where $v = \frac{\partial ||u||}{\partial u}\Big|_{u=\widetilde{w}}$. Note that even for a unit-norm constraint with any arbitrary norm $||\cdot||$, it is still the L^2 norm of v which appears in the expression of the tangent gradient, due to the orthogonal projection being performed in Euclidean space.

Interestingly, for the L^2-norm constraint, we see from Eq. 13 that the gradient $\nabla\widetilde{\mathcal{I}}$ is equal up to a scaling factor to the tangent gradient of $\mathcal{I}$ taken at point $\widetilde{w}$. Indeed, we then have $v = \widetilde{w}$ and $||v||_2 = 1$, and thus

$$\nabla \widetilde{\mathcal{I}}(w) = \frac{1}{||w||}\nabla^{(T)}\mathcal{I}(\widetilde{w}). \tag{15}$$

4.3 Relation to the natural gradient

The natural gradient [15] of a function $\mathcal{I}$ defined on a manifold $\mathcal{S}$ is a modification of its standard gradient according to the local curvature of the parameter space. It can be obtained from the standard gradient $\nabla\mathcal{I}$ as

$$\nabla^{(N)}\mathcal{I}(s) = \mathbf{G}^{-1}(s)\nabla\mathcal{I}(s), \forall s \in \mathcal{S}, \tag{16}$$

where $\mathcal{S}$ is here the constraint manifold, and $\mathbf{G}$ is the Riemannian metric tensor of the constraint manifold at s.

In the particular case of the L^2-norm constraint, the tangent gradient of the original objective function $\mathcal{I}$ can be shown to be equal, on the unit-norm surface, to its natural gradient:

$$\nabla^{(N)}\mathcal{I}(\widetilde{w}) = \nabla^{(T)}\mathcal{I}(\widetilde{w}), \quad \forall \widetilde{w} \text{ s.t. } ||\widetilde{w}||_2 = 1. \tag{17}$$

However, while this is reported as a fact in the literature [17], to our knowledge, no proof is available. We give a detailed proof in Appendix A.

Combining with Eq. 15, we obtain:

$$\nabla\widetilde{\mathcal{I}}(\widetilde{w}) = \nabla^{(T)}\mathcal{I}(\widetilde{w}) = \nabla^{(N)}\mathcal{I}(\widetilde{w}), \quad \forall \widetilde{w} \text{ s.t. } ||\widetilde{w}||_2 = 1. \tag{18}$$

Altogether, starting from a point $\widetilde{w}$ on the unit-norm manifold for L^2, the gradient of the objective function $\widetilde{\mathcal{I}}$ with explicitly normalized parameters is equal to the natural gradient of the original objective function $\mathcal{I}$ as defined on the unit-norm manifold.

5 Update equations for SNMF with β-divergence

We come back to the derivation of the update equations for (7). The update equations for $\mathbf{H}$ are unchanged, except that they now use the normalized version $\widetilde{\mathbf{W}}$ of $\mathbf{W}$. Let us derive multiplicative update equations for $\mathbf{W}$ by splitting the expression for the gradient into positive and negative parts, similarly to [20].

We first compute the gradient of $D_\beta(\mathbf{M} \mid \mathbf{WH})$ with respect to $\mathbf{W}_i$:

$$\nabla_{\mathbf{W}_i} D_\beta(\mathbf{M} \mid \mathbf{WH}) = \Big(\big(\mathbf{WH}\big)^{\beta-2} \otimes \big(\mathbf{WH} - \mathbf{M}\big)\Big)\mathbf{H}_i^\intercal, \tag{19}$$

where the $\otimes$ product and the operation of raising to an exponent are considered element-wise. To avoid clutter, we will use the notation $\mathbf{\Lambda} = \widetilde{\mathbf{W}}\mathbf{H}$. From there and the result of Section 4.1, we can compute the gradient of $D_\beta(\mathbf{M} \mid \widetilde{\mathbf{W}}\mathbf{H})$ with respect to $\mathbf{W}_i$:

$$\nabla_{\mathbf{W}_i} D_\beta(\mathbf{M} \mid \mathbf{\Lambda}) = \frac{1}{||\mathbf{W}_i||}\Big(\mathrm{Id} - \widetilde{\mathbf{W}}_i\widetilde{\mathbf{W}}_i^\intercal\Big)\Big(\mathbf{\Lambda}^{\beta-2} \otimes \big(\mathbf{\Lambda} - \mathbf{M}\big)\Big)\mathbf{H}_i^\intercal. \tag{20}$$

We split the gradient into a positive part $[\nabla_{\mathbf{W}_i} D_\beta(\mathbf{M} \mid \mathbf{\Lambda})]_+$ and a negative part $[\nabla_{\mathbf{W}_i} D_\beta(\mathbf{M} \mid \mathbf{\Lambda})]_-$:

$$[\nabla_{\mathbf{W}_i} D_\beta(\mathbf{M} \mid \mathbf{\Lambda})]_+ = \frac{1}{||\mathbf{W}_i||}\Big(\mathbf{\Lambda}^{\beta-1} + \widetilde{\mathbf{W}}_i\widetilde{\mathbf{W}}_i^\intercal\Big(\mathbf{\Lambda}^{\beta-2} \otimes \mathbf{M}\Big)\Big)\mathbf{H}_i^\intercal \tag{21}$$

$$[\nabla_{\mathbf{W}_i} D_\beta(\mathbf{M} \mid \mathbf{\Lambda})]_- = \frac{1}{||\mathbf{W}_i||}\Big(\mathbf{\Lambda}^{\beta-2}\mathbf{M} + \widetilde{\mathbf{W}}_i\widetilde{\mathbf{W}}_i^\intercal\mathbf{\Lambda}^{\beta-1}\Big)\mathbf{H}_i^\intercal \tag{22}$$

We thus obtain the following multiplicative updates for $\mathbf{W}_i$:

$$\mathbf{W}_i \leftarrow \mathbf{W}_i \otimes \frac{[\nabla_{\mathbf{W}_i} D_\beta(\mathbf{M} \mid \mathbf{\Lambda})]_-}{[\nabla_{\mathbf{W}_i} D_\beta(\mathbf{M} \mid \mathbf{\Lambda})]_+} \tag{23}$$

$$= \mathbf{W}_i \otimes \frac{\Big(\mathbf{\Lambda}^{\beta-2} \otimes \mathbf{M} + \widetilde{\mathbf{W}}_i\widetilde{\mathbf{W}}_i^\intercal\mathbf{\Lambda}^{\beta-1}\Big)\mathbf{H}_i^\intercal}{\Big(\mathbf{\Lambda}^{\beta-1} + \widetilde{\mathbf{W}}_i\widetilde{\mathbf{W}}_i^\intercal\Big(\mathbf{\Lambda}^{\beta-2} \otimes \mathbf{M}\Big)\Big)\mathbf{H}_i^\intercal}. \tag{24}$$

Note that this update can be computed efficiently for the whole matrix using the following equivalent form:

$$\mathbf{W} \leftarrow \mathbf{W} \otimes \frac{\Big(\mathbf{\Lambda}^{\beta-2} \otimes \mathbf{M}\Big)\mathbf{H}^\intercal + \widetilde{\mathbf{W}} \otimes \Big(\mathbf{1}\mathbf{1}^\intercal\big(\widetilde{\mathbf{W}} \otimes \big(\mathbf{\Lambda}^{\beta-1}\mathbf{H}^\intercal\big)\big)\Big)}{\mathbf{\Lambda}^{\beta-1}\mathbf{H}^\intercal + \widetilde{\mathbf{W}} \otimes \Big(\mathbf{1}\mathbf{1}^\intercal\Big(\widetilde{\mathbf{W}} \otimes \big((\mathbf{\Lambda}^{\beta-2} \otimes \mathbf{M})\mathbf{H}^\intercal\big)\Big)\Big)}, \tag{25}$$

where $\mathbf{1}$ is a column vector with all elements equal to 1. The operation $\widetilde{\mathbf{W}} \otimes (\mathbf{1}\mathbf{v}^\intercal)$, where $\mathbf{v}$ is a column vector, can be carried out efficiently in Matlab using a command such as

Algorithm 1 NMF+S: NMF with Sparsity, half-baked

Inputs: $\mathbf{M}$, R, $\beta \geq 0$, $\mu \geq 0$
$\mathbf{W}$ initialized randomly or by sampling from the data, $\mathbf{H}$ initialized randomly
$\mathbf{H} \leftarrow \widetilde{\mathbf{H}}$, where $\widetilde{\mathbf{H}} = \left[\|\mathbf{w}_1\| \, \mathbf{h}_1; \cdots ; \|\mathbf{w}_R\| \, \mathbf{h}_R\right]$
$\mathbf{W} \leftarrow \widetilde{\mathbf{W}}$
$\mathbf{\Lambda} = \mathbf{WH}$
repeat
  $\mathbf{H} \leftarrow \mathbf{H} \otimes \dfrac{\mathbf{W}^\intercal(\mathbf{M} \otimes \mathbf{\Lambda}^{\beta-2})}{\mathbf{W}^\intercal \mathbf{\Lambda}^{\beta-1} + \mu}$
  $\mathbf{\Lambda} = \mathbf{WH}$
  $\mathbf{W} \leftarrow \mathbf{W} \otimes \dfrac{(\mathbf{\Lambda}^{\beta-2} \otimes \mathbf{M})\mathbf{H}^\intercal}{\mathbf{\Lambda}^{\beta-1}\mathbf{H}^\intercal}$
  $\mathbf{H} \leftarrow \widetilde{\mathbf{H}}$, $\mathbf{W} \leftarrow \widetilde{\mathbf{W}}$
  $\mathbf{\Lambda} = \mathbf{WH}$
until convergence
return $\mathbf{W}, \mathbf{H}$

Algorithm 2 SNMF: Sparse NMF, well done

Inputs: $\mathbf{M}$, R, $\beta \geq 0$, $\mu \geq 0$
$\mathbf{W}$ initialized randomly or by sampling from the data, $\mathbf{H}$ initialized randomly
$\mathbf{W} \leftarrow \widetilde{\mathbf{W}}$
$\mathbf{\Lambda} = \mathbf{WH}$
repeat
  $\mathbf{H} \leftarrow \mathbf{H} \otimes \dfrac{\widetilde{\mathbf{W}}^\intercal(\mathbf{M} \otimes \mathbf{\Lambda}^{\beta-2})}{\widetilde{\mathbf{W}}^\intercal \mathbf{\Lambda}^{\beta-1} + \mu}$
  $\mathbf{\Lambda} = \mathbf{WH}$
  $\mathbf{W} \leftarrow \mathbf{W} \otimes \dfrac{\left(\mathbf{\Lambda}^{\beta-2} \otimes \mathbf{M}\right)\mathbf{H}^\intercal + \widetilde{\mathbf{W}} \otimes \left(\mathbf{1}\mathbf{1}^\intercal(\widetilde{\mathbf{W}} \otimes (\mathbf{\Lambda}^{\beta-1}\mathbf{H}^\intercal))\right)}{\mathbf{\Lambda}^{\beta-1}\mathbf{H}^\intercal + \widetilde{\mathbf{W}} \otimes \left(\mathbf{1}\mathbf{1}^\intercal\left(\widetilde{\mathbf{W}} \otimes \left((\mathbf{\Lambda}^{\beta-2} \otimes \mathbf{M})\mathbf{H}^\intercal\right)\right)\right)}$
  $\mathbf{W} \leftarrow \widetilde{\mathbf{W}}$
  $\mathbf{\Lambda} = \mathbf{WH}$
until convergence
return $\mathbf{W}, \mathbf{H}$

`bsxfun(@times,Wtilde,v')`. For convenience of computation, we renormalize $\mathbf{W}$ at the end of each iteration, but we do not rescale $\mathbf{H}$ accordingly. Because only $\widetilde{\mathbf{W}}$ appears in the objective function, this does not change the value of the objective function, and it avoids having to keep both a normalized and an unnormalized version of $\mathbf{W}$. The algorithm would lead to exactly the same $\widetilde{\mathbf{W}}$ if we did not normalize $\mathbf{W}$ and performed the iterative updates. The training procedure with SNMF is summarized in Algorithm 2. The source code for an efficient MATLAB® implementation is publicly available for research-only purposes at the following address: `ftp://ftp.merl.com/pub/leroux/sparseNMF.zip`.

As a comparison, for NMF+S, the updates for $\mathbf{W}$ are simply:

$$\mathbf{W} \leftarrow \mathbf{W} \otimes \frac{\left(\mathbf{\Lambda}^{\beta-2} \otimes \mathbf{M}\right)\mathbf{H}^\intercal}{\mathbf{\Lambda}^{\beta-1}\mathbf{H}^\intercal}, \tag{26}$$

followed by re-normalization of $\mathbf{W}$ and the corresponding rescaling of $\mathbf{H}$. The training procedure with NMF+S is summarized in Algorithm 1.

6 Source separation using NMF

NMF has been a very popular algorithm in the audio community, and in particular it is commonly used for challenging single-channel audio source separation tasks, such as speech enhancement in the presence of non-stationary noises [10, 21]. In this context, the basic idea is to represent the

features of the sources via sets of basis functions and their activation coefficients, one set per source. Mixtures of signals are then analyzed using the concatenated sets of basis functions, and each source is reconstructed using its corresponding activations and basis set. In order to accurately represent each source, the sets of basis functions need to be large, counting potentially hundreds or thousands of basis functions. While vanilla NMF can be useful to obtain low-rank approximations, it leads to trivial solutions when used with a number of basis functions comparable to or higher than the dimension of the feature space in which it operates.

In the context of audio source separation, NMF operates on a matrix of F-dimensional non-negative spectral features, usually the power or magnitude spectrogram of the mixture, $\mathbf{M} = [\mathbf{m}_1 \cdots \mathbf{m}_T]$, where T is the number of frames and $\mathbf{m}_t \in \mathbb{R}_+^F$, $t = 1, \ldots, T$ are obtained by short-time Fourier analysis of the time-domain signal. With L sources, each source $l \in \{1, \ldots, L\}$ is represented using a matrix containing R_l non-negative basis column vectors, $\mathbf{W}^l = \{\mathbf{w}_r^l\}_{r=1}^{R_l}$, multiplied by a matrix of activation column vectors $\mathbf{H}^l = \{\mathbf{h}_t^l\}_{t=1}^T$, for each time t. The rth row of $\mathbf{H}^l$ contains the activations for the corresponding basis $\mathbf{w}_r^l$ at each time t. From this, a factorization

$$\mathbf{M} \approx \sum_l \mathbf{S}^l \approx [\mathbf{W}^1 \cdots \mathbf{W}^S][\mathbf{H}^1; \cdots; \mathbf{H}^S] = \mathbf{W}\mathbf{H} \tag{27}$$

is obtained, where we use the notation $[\mathbf{a}; \mathbf{b}]$ for $[\mathbf{a}^\intercal \mathbf{b}^\intercal]^\intercal$. An approach related to Wiener filtering is typically used to reconstruct each source while ensuring that the source estimates sum to the mixture:

$$\hat{\mathbf{S}}^l = \frac{\mathbf{W}^l\mathbf{H}^l}{\sum_l \mathbf{W}^l\mathbf{H}^l} \otimes \mathbf{M}, \tag{28}$$

where $\otimes$ denotes element-wise multiplication and the quotient line element-wise division. In our study, all $\mathbf{W}^l$ are learned or obtained in advance from training data, using one of the algorithms of Section 3.1. At test time, only the activation matrices $\hat{\mathbf{H}} = [\hat{\mathbf{H}}^1; \cdots; \hat{\mathbf{H}}^S]$ are estimated, using the update equations (4), so as to find a (local) minimum of (3). This is called *supervised NMF* [22]. In the supervised case, the activations for each frame are independent from the other frames ($\mathbf{m}_t \approx \sum_l \mathbf{W}^l \mathbf{h}_t^l$). Thus, source separation can be performed on-line and with latency corresponding to the window length plus the computation time to obtain the activations for one frame [21].

Since sources often have similar characteristics in the short-term observations (such as unvoiced phonemes and broadband noise, or voiced phonemes and music), it seems beneficial to use information from multiple time frames. In our study, this is done by stacking features from multiple contiguous time frames into 'supervectors': the observation $\mathbf{m}'_t$ at time t corresponds to the observations $[\mathbf{m}_{t-T_\mathrm{L}}; \cdots; \mathbf{m}_t; \cdots; \mathbf{m}_{t+T_\mathrm{R}}]$ where T_L and T_R are the left and right context sizes. Missing observations at the beginning and end of the data are replaced by copying the first and last observations, i.e., $\mathbf{m}_{t:t<0} := \mathbf{m}_1$ and $\mathbf{m}_{t:t>T} := \mathbf{m}_T$. Analogously, each basis element $\mathbf{w}'^l_k$ will model a sequence of spectra, stacked into a column vector. For readability, we omit the $'$ from the matrix names and assume that all features correspond to stacked column vectors of short-time spectra.

7 Experiments and Results

The algorithms are evaluated on the corpus of the 2nd CHiME Speech Separation and Recognition Challenge, which is publicly available[1]. The task is to separate speech from noisy and reverberated mixtures. The noise was recorded in a home environment with mostly non-stationary noise sources such as children, household appliances, television, radio, etc. Training, development, and test sets of noisy mixtures along with noise-free reference signals are created from the Wall Street Journal (WSJ-0) corpus of read speech and a corpus of training noise recordings. The dry speech recordings are convolved with room impulse responses from the same environment where the noise corpus is recorded. The training set consists of 7 138 utterances at six signal-to-noise ratios (SNRs) from -6 to 9 dB, in steps of 3 dB. The development and test sets consist of 410 and 330 utterances at each of these SNRs, for a total of 2 460 / 1 980 utterances. Our evaluation measure for speech separation is source-to-distortion ratio (SDR) [23]. By construction of the WSJ-0 corpus, our evaluation is speaker-independent. Furthermore, the background noise in the development and test set is disjoint from the training noise, and a different room impulse response is used to convolve the dry utterances.

[1] http://spandh.dcs.shef.ac.uk/chime_challenge/ – as of Feb. 2014

7.1 Feature extraction

Each feature vector (in the mixture, source, and reconstructed source spectrograms as well as the basis vectors) covers nine consecutive frames ($T_{\mathrm{L}} = 8$, $T_{\mathrm{R}} = 0$) obtained as short-time Fourier spectral magnitudes, using 25 ms window size, 10 ms window shift, and the square root of the Hann window. Since no information from the future is used ($T_{\mathrm{R}} = 0$), the observation features ($\mathbf{m}_t$) can be extracted on-line. In analogy to the features in $\mathbf{M}$, each column of $\hat{\mathbf{S}}^l$ corresponds to a sliding window of consecutive reconstructed frames. Only the last frame in each sliding window is reconstructed, which leads to an on-line algorithm.

7.2 Practical implementation using exemplar-based and sparse NMF

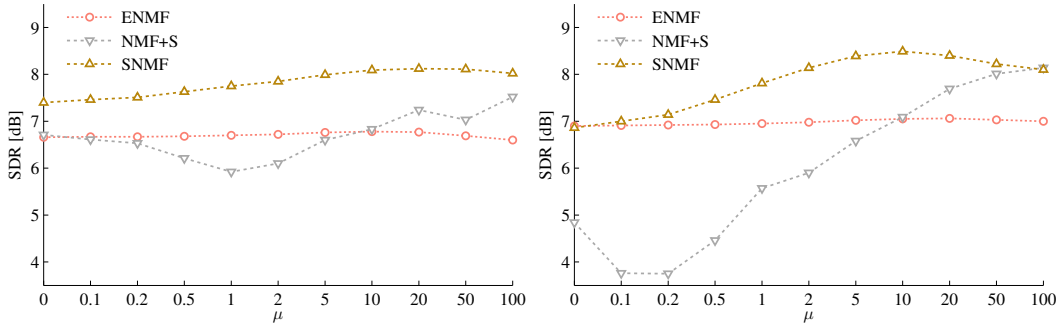
We use the same number R of basis vectors for speech and noise ($R^1 = R^2 = R$). We run an experiment for $R = 100$ and $R = 1000$. The maximum number of iterations at test time is set to $Q = 25$ based on the trade-off between SDR and complexity – running NMF until convergence increased SDR only by about 0.1 dB SDR in preliminary experiments. At training time, we use up to $Q = 100$ iterations. As described above, we consider three different approaches to obtain NMF basis functions. In ENMF, the set of basis functions $\mathbf{W}$ corresponds to $R^1 + R^2 = 2R$ randomly selected spectral patches of speech and noise, each spanning $T_{\mathrm{L}} + 1 = 9$ frames, from the isolated CHiME speech and background noise training sets. For SNMF and NMF+S, basis training is performed separately for the speech and the noise: setting $\mathbf{S}^1$ to the spectrograms of the concatenated noise-free CHiME training set and $\mathbf{S}^2$ to those of the corresponding background noise in the multi-condition training set yields bases $\mathbf{W}^l$, $l = 1, 2$. Due to space complexity, we use only 10 % of the training utterances. We initialize $\mathbf{W}$ using the exemplar bases. We found that this provided fast convergence of the objective especially for large values of μ. The update procedure for NMF+S follows Algorithm 1, while that for SNMF follows Algorithm 2. Note that, in SNMF, even though normalization is explicitly taken into account in the objective function, we still re-normalize $\mathbf{W}$ at the end of every iteration so that the only difference with NMF+S is limited to the update equations for $\mathbf{W}$.

7.3 Results on the CHiME development and test set

We compare the three approaches for three different settings of the β-divergence: $\beta = 0$ (IS divergence), $\beta = 1$ (KL divergence), and $\beta = 2$ (squared Euclidean distance). For each divergence, we investigate two settings of the basis size R, $R \in \{100, 1000\}$, and multiple settings of the sparsity weight μ in (3), $\mu \in \{0, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100\}$. At test time, all settings are explored using the bases trained using the corresponding setting for SNMF and NMF+S, and the exemplar bases for ENMF.

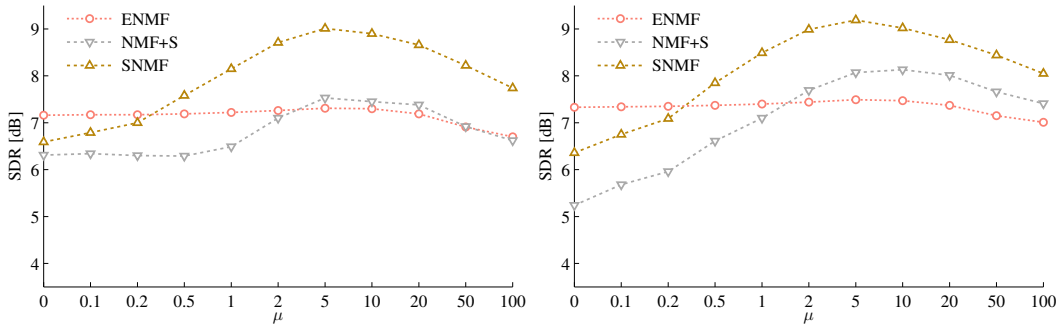
Figure 2 compares the average SDR obtained on the CHiME development set by using ENMF, NMF+S and SNMF, for various sparsity parameters as well as basis sizes R, for the IS distance ($\beta = 0$). Figure 3 and Figure 4 show the same comparison for the KL divergence ($\beta = 1$) and the Euclidean distance ($\beta = 2$), respectively. It can be seen that SNMF outperforms ENMF for all sparsity values for the IS distance, and for higher sparsity values ($\mu \geq 0.5$) for the KL divergence and the Euclidean distance. One can also see that SNMF consistently and significantly outperforms its ad hoc counterpart NMF+S for all divergence functions, number of basis functions, and sparsity weights, except for very high sparsity weights ($\mu \geq 50$) for the Euclidean distance; note however that the performance for such sparsity weights is much lower than the optimal performance obtained by SNMF, which is reached consistently for sparsity weights around 5 to 10 in our experiments. It is also interesting to note that the performance of NMF+S is generally either comparable to or worse than that of the exemplar-based basis selection method ENMF, which has the extra advantage that it does not require any training. This is arguably one of the reasons why sampling the basis functions instead of training them has been so popular in NMF-based works.

Overall, the best performance on the development set is obtained by SNMF, for the KL-divergence with $R = 1000$ and $\mu = 5$, with an SDR of 9.20 dB. The best performance for ENMF is obtained for the KL divergence with $R = 1000$ and $\mu = 5$, with an SDR of 7.49 dB, while that of NMF+S is obtained for the Itakura-Saito distance with $R = 1000$ and $\mu = 20$, with an SDR of 7.69 dB.



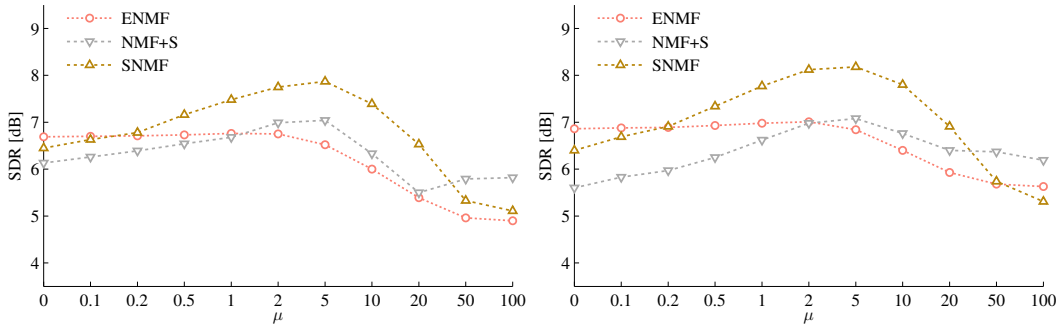
(a) Results for $\beta = 0$, $R = 100$ (b) Results for $\beta = 0$, $R = 1000$

Figure 2: Average SDR obtained with the Itakura-Saito divergence ($\beta = 0$) for various sparsity weights μ on the CHiME Challenge (WSJ-0) development set.



(a) Results for $\beta = 1$, $R = 100$ (b) Results for $\beta = 1$, $R = 1000$

Figure 3: Average SDR obtained with the Kullback-Leibler divergence ($\beta = 1$) for various sparsity weights μ on the CHiME Challenge (WSJ-0) development set.



(a) Results for $\beta = 2$, $R = 100$ (b) Results for $\beta = 2$, $R = 1000$

Figure 4: Average SDR obtained with the Euclidean distance ($\beta = 2$) for various sparsity weights μ on the CHiME Challenge (WSJ-0) development set.

Table 1 shows the results on the CHiME test set by ENMF, NMF+S and SNMF for the KL divergence, using $\mu = 5$ as tuned on the development set, for $R = 1000$. The results mirror those obtained on the development set.

7.4 Influence of initialization

We compare here various procedures to initialize the basis functions and activations before training. The default procedure, referred to as eW-rH, is the one used in the above experiments, where the basis functions $\mathbf{W}$ are obtained as the exemplar bases, and $\mathbf{H}$ is randomly initialized with uniform

SDR [dB]	Input SNR [dB]						
	-6	-3	0	3	6	9	Avg.
Noisy	-2.27	-0.58	1.66	3.40	5.20	6.60	2.34
ENMF	3.01	5.58	7.60	9.58	11.79	13.67	8.54
NMF+S	3.60	5.98	7.58	9.19	10.95	12.16	8.24
SNMF	5.48	7.53	9.19	10.88	12.89	14.61	10.10

Table 1: Source separation performance on the CHiME Challenge (WSJ-0) test set using KL-divergence, $\mu = 5$ and $R = 1000$.

SDR [dB]	Input SNR [dB]						
	-6	-3	0	3	6	9	Avg.
NMF+S rW-rH	-0.24	2.63	5.57	7.48	9.54	11.91	6.15
NMF+S eW-rH	2.91	5.40	7.72	9.12	10.80	12.47	8.07
NMF+S eW-oH	2.48	5.00	7.46	8.99	10.80	12.71	7.91
SNMF rW-rH	4.66	6.82	8.78	10.18	11.86	13.69	9.33
SNMF eW-rH	4.37	6.59	8.66	10.06	11.81	13.67	9.19
SNMF eW-oH	3.63	5.98	8.21	9.66	11.48	13.43	8.73

Table 2: Influence of initialization of $\mathbf{W}$ and $\mathbf{H}$ during training on source separation performance, measured on the CHiME Challenge (WSJ-0) development set using KL-divergence, $\mu = 5$ and $R = 1000$. "rW-rH": initialize both $\mathbf{W}$ and $\mathbf{H}$ randomly; "eW-rH": initialize $\mathbf{W}$ using ENMF and $\mathbf{H}$ randomly; "eW-oH": initialize $\mathbf{W}$ using ENMF, then initialize $\mathbf{H}$ by optimizing it with fixed $\mathbf{W}$.

distribution on the open interval $(0, 1)$. We consider two other initialization procedures: in one, referred to as rW-rH, both $\mathbf{W}$ and $\mathbf{H}$ are randomly initialized; in the other, referred to as eW-oH, we start with the same initialization as eW-rH, then optimize $\mathbf{H}$ with fixed $\mathbf{W}$ until convergence, after which both $\mathbf{W}$ and $\mathbf{H}$ are optimized as usual. The intent here is to look for the best initialization strategy. We ran these experiments in a single setting, using the KL divergence, with $\mu = 5$ and $K = 1000$. Results are reported in Table 2.

Interestingly, optimizing $\mathbf{H}$ (eW-oH) led to the worst results for SNMF, while completely random initialization (rW-rH) led to the best results, only outperforming our default setup (eW-rH) by a small margin. This hints at the tendency of the algorithm to get stuck into local minima, which is expected with multiplicative update type algorithms for NMF. On the other hand, these results show that SNMF can robustly estimate good-performing basis functions from a random initialization. Such is not the case for NMF+S, where rW-rH led to the worst results: one explanation is that the algorithm being wrong in the first place, it benefits from being initialized with the somewhat relevant basis functions obtained with ENMF.

8 Conclusion

We presented a thorough comparison of three methods for obtaining NMF basis functions from training data: ENMF, which relies on sampling from the data; NMF+S, which attempts to perform unsupervised NMF with sparsity using a simple ad-hoc implementation; and SNMF, which actually optimizes a sparse NMF objective function. We showed that, on a large and challenging speech separation task, SNMF does lead to better basis functions than ENMF, while NMF+S leads to poorer results than either. Since NMF+S is so temptingly simple to implement, it may have convinced some researchers that using samples was better than learning the bases. Exemplar-based methods may still be useful in situations where training complexity is an issue. However, when the best performance is desired, SNMF should be preferred.

Appendix A Proof of equality of the tangent gradient and the natural gradient for the L^2-norm constraint

The natural gradient [15] is a modification of the standard gradient according to the local curvature of the parameter space. It can be obtained from the standard gradient as

$$\nabla^{(N)}\mathcal{I}(s) = \mathbf{G}^{-1}(s)\nabla\mathcal{I}(s), \forall s \in \mathcal{S}, \tag{29}$$

where $\mathcal{S}$ is a manifold, and $\mathbf{G}$ is the Riemannian metric tensor of the manifold at s. The metric tensor at point s defines the dot product on the tangent space at s. For $\mathbb{R}^n$ with Cartesian coordinates, the metric tensor is simply the identity matrix, and the dot product between vectors is defined as usual. Note that the definition of the manifold and $\mathbf{G}$ encompasses the coordinate system in which s is expressed. For example, $\mathbb{R}^2 \setminus \{(0,0)\}$ with Cartesian coordinates is a different manifold from $\mathbb{R}^2 \setminus \{(0,0)\}$ with polar coordinates. In order to obtain the natural gradient as expressed in another coordinate system, one needs to perform a change of variables. This will be our strategy here: compute the metric tensor using a spherical coordinate system, in which it is easier to define the unit-norm manifold, and use a change of variables to express the natural gradient in the Cartesian coordinate system. From here on, we assume that $\mathcal{S}$ is the unit L^2-norm constraint manifold in $\mathbb{R}^n$, or in other words the Cartesian n-sphere of radius 1.

First, we derive the metric tensor of the n-sphere. We denote by f the mapping from the spherical coordinate system in n dimensions to the Cartesian coordinates, $f : (r, \phi_1, \dots, \phi_{n-1}) \mapsto (x_1, \dots, x_n)$ such that $f_i(r, \phi_1, \dots, \phi_{n-1}) = x_i$ with

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} r\cos\phi_1 \\ r\sin\phi_1\cos\phi_2 \\ \vdots \\ r\sin\phi_1\cdots\sin\phi_{i-1}\cos\phi_i \\ \vdots \\ r\sin\phi_1\cdots\cdots\sin\phi_{n-2}\cos\phi_{n-1} \\ r\sin\phi_1\cdots\cdots\sin\phi_{n-2}\sin\phi_{n-1} \end{bmatrix} \tag{30}$$

Then, the metric tensor in the spherical coordinate system $\mathbf{G}^{(r,\boldsymbol{\phi})}$ can be obtained from that in the Cartesian coordinate system $\mathbf{G}^{(\boldsymbol{x})}$ using a (reverse) change of coordinates:

$$\mathbf{G}^{(r,\boldsymbol{\phi})} = (D_{r,\boldsymbol{\phi}}f)^\intercal \mathbf{G}^{(\boldsymbol{x})} D_{r,\boldsymbol{\phi}}f, \tag{31}$$

where $D_{r,\boldsymbol{\phi}}f$ is the Jacobian matrix of f. For convenience, we introduce the notation:

$$D_{r,\boldsymbol{\phi}}f = [\ \nabla_r f \mid \quad D_{\boldsymbol{\phi}}f \quad] = [\ \nabla_r \boldsymbol{x} \mid \quad D_{\boldsymbol{\phi}}\boldsymbol{x} \quad]. \tag{32}$$

As $\mathbf{G}^{(\boldsymbol{x})}$ is equal to Id, we obtain:

$$\mathbf{G}^{(r,\boldsymbol{\phi})} = (D_{r,\boldsymbol{\phi}}f)^\intercal D_{r,\boldsymbol{\phi}}f. \tag{33}$$

Let us explicitly compute $D_{r,\boldsymbol{\phi}}f$. First, it is easy to see that $\nabla_r f = \frac{1}{r}\boldsymbol{x}$, and

$$\nabla_{\phi_i} f = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -r\sin\phi_1\cdots\sin\phi_{i-1}\sin\phi_i \\ r\sin\phi_1\cdots\sin\phi_{i-1}\cos\phi_i\cos\phi_{i+1} \\ \vdots \\ r\sin\phi_1\cdots\sin\phi_{i-1}\cos\phi_i\sin\phi_{i+1}\cdots\sin\phi_{n-1}\cos\phi_{n-1} \\ r\sin\phi_1\cdots\sin\phi_{i-1}\cos\phi_i\sin\phi_{i+1}\cdots\sin\phi_{n-1}\sin\phi_{n-1} \end{bmatrix} \begin{matrix} \\ \\ \\ \leftarrow\text{i-th row} \\ \\ \\ \\ \\ \end{matrix},$$

where the cases $i = 1$ and $i = n - 1$ can be easily inferred from the above.

We can see that

$$G^{(r,\boldsymbol{\phi})}_{rr} = 1, \tag{34}$$

$$G^{(r,\boldsymbol{\phi})}_{\phi_i\phi_i} = r^2 \prod_{l=1}^{i-1} \sin^2 \phi_l, \tag{35}$$

$$G^{(r,\boldsymbol{\phi})}_{r\phi_i} = 0, \tag{36}$$

$$G^{(r,\boldsymbol{\phi})}_{\phi_i\phi_j} = 0, \quad i \neq j. \tag{37}$$

The computation of $G^{(r,\boldsymbol{\phi})}_{rr}$ is straightforward. For $G^{(r,\boldsymbol{\phi})}_{\phi_i\phi_i}$, the result can be obtain by noticing that

$$G^{(r,\boldsymbol{\phi})}_{\phi_i\phi_i} = r^2 \Big(\prod_{l=1}^{i-1} \sin^2 \phi_l \Big) (\sin^2 \phi_i + \cos^2 \phi_i \cdot \eta_{i+1}) \tag{38}$$

where η_l is defined by backward recurrence using

$$\eta_l = (\cos^2 \phi_l + \sin^2 \phi_l \cdot \eta_{l+1}), \text{ and } \eta_{n-1} = \cos^2 \phi_{n-1} + \sin^2 \phi_{n-1} = 1. \tag{39}$$

One can easily see that $\eta_l = 1, \forall l \in i+1, \ldots, n-2$ as well, and thus that the factor $(\sin^2 \phi_i + \cos^2 \phi_i \cdot \eta_{i+1})$ simplifies to 1. We now compute $G^{(r,\boldsymbol{\phi})}_{r\phi_i}$:

$$G^{(r,\boldsymbol{\phi})}_{r\phi_i} = r \Big(\prod_{l=1}^{i-1} \sin^2 \phi_l \Big) (-\cos \phi_i \sin \phi_i + \sin \phi_i \cos \phi_i \cdot \eta_{i+1}). \tag{40}$$

As seen above, $\eta_{j+1} = 1$, and the right-hand term thus cancels out. Similarly, we compute $G^{(r,\boldsymbol{\phi})}_{\phi_i\phi_j}$ with $i < j$ (the case of $j > i$ is symmetric):

$$G^{(r,\boldsymbol{\phi})}_{\phi_i\phi_j} = r^2 \Big(\prod_{\substack{l=1 \\ l\neq i}}^{j-1} \sin^2 \phi_l \Big) (-(\cos \phi_i \cos \phi_j)(\sin \phi_i \sin \phi_j) + (\cos \phi_i \sin \phi_j)(\sin \phi_i \cos \phi_j)\eta_{j+1}), \tag{41}$$

which again cancels out. Note that the above result is only given for the sake of completeness, and we actually do not use the value of $G^{(r,\boldsymbol{\phi})}_{\phi_i\phi_j}$ for $i \neq j$ in the remainder.

Altogether, we can write

$$\mathbf{G}^{(r,\boldsymbol{\phi})} = \left[\begin{array}{c|ccc} 1 & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & \mathbf{G}^{(\boldsymbol{\phi})} & \\ 0 & & & \end{array}\right] \tag{42}$$

The quantity we are interested in is the inverse of the metric tensor of the unit-norm manifold, $(\mathbf{G}^{(\boldsymbol{\phi})})^{-1}$, expressed in Cartesian coordinates, i.e., $(D_{\boldsymbol{\phi}} f)(\mathbf{G}^{(\boldsymbol{\phi})})^{-1}(D_{\boldsymbol{\phi}} f)^\intercal$. But noticing that

$$(D_{r,\boldsymbol{\phi}} f)(\mathbf{G}^{(r,\boldsymbol{\phi})})^{-1}(D_{r,\boldsymbol{\phi}} f)^\intercal = \mathrm{Id}, \tag{43}$$

we obtain

$$\mathrm{Id} = \left[\begin{array}{c|c} \vdots & \\ \frac{1}{r}\boldsymbol{x} & D_{\boldsymbol{\phi}} f \\ \vdots & \end{array}\right] \left[\begin{array}{c|ccc} 1 & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & \mathbf{G}^{(\boldsymbol{\phi})} & \\ 0 & & & \end{array}\right]^{-1} \left[\begin{array}{c|c} \vdots & \\ \frac{1}{r}\boldsymbol{x} & D_{\boldsymbol{\phi}} f \\ \vdots & \end{array}\right]^\intercal \tag{44}$$

$$= \left[\begin{array}{c|c} \vdots & \\ \frac{1}{r}\boldsymbol{x} & (D_{\boldsymbol{\phi}} f)(\mathbf{G}^{(\boldsymbol{\phi})})^{-1} \\ \vdots & \end{array}\right] \left[\begin{array}{c|c} \vdots & \\ \frac{1}{r}\boldsymbol{x} & D_{\boldsymbol{\phi}} f \\ \vdots & \end{array}\right]^\intercal \tag{45}$$

$$= \frac{1}{r^2}\boldsymbol{x}\boldsymbol{x}^\intercal + (D_{\boldsymbol{\phi}} f)(\mathbf{G}^{(\boldsymbol{\phi})})^{-1}(D_{\boldsymbol{\phi}} f)^\intercal \tag{46}$$

which leads to:

$$(D_{\boldsymbol{\phi}} f)(\mathbf{G}^{(\boldsymbol{\phi})})^{-1}(D_{\boldsymbol{\phi}} f)^{\intercal} = \mathrm{Id} - \frac{1}{r^2}\boldsymbol{x}\boldsymbol{x}^{\intercal}. \tag{47}$$

For a point $\boldsymbol{x}$ on the unit-norm surface, we recognize the projection operator used to compute the tangent gradient in (14), and we thus obtain:

$$\nabla^{(N)}\mathcal{I}(s) = (D_{\boldsymbol{\phi}} f)(\mathbf{G}^{(\boldsymbol{\phi})})^{-1}(D_{\boldsymbol{\phi}} f)^{\intercal}\nabla\mathcal{I}(s) = \nabla^{(T)}\mathcal{I}(s), \forall s \in \mathcal{S}. \tag{48}$$

Appendix B Source code for SNMF

B.1 `example.m`

```
% Example script for Sparse NMF with beta-divergence distortion function
% and L1 penalty on the activations.
%
% If you use this code, please cite:
% J. Le Roux, J. R. Hershey, F. Weninger,
% "Sparse NMF - half-baked or well done?,"
% MERL Technical Report, TR2015-023, March 2015
%   @TechRep{LeRoux2015mar,
%     author = {{Le Roux}, J. and Hershey, J. R. and Weninger, F.},
%     title = {Sparse {NMF} - half-baked or well done?},
%     institution = {Mitsubishi Electric Research Labs (MERL)},
%     number = {TR2015-023},
%     address = {Cambridge, MA, USA},
%     month = mar,
%     year = 2015
%   }
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Copyright (C) 2015 Mitsubishi Electric Research Labs (Jonathan Le Roux,
%                                                  Felix Weninger, John R. Hershey)
%   Apache 2.0  (http://www.apache.org/licenses/LICENSE-2.0)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% You need to provide a non-negative matrix v to be factorized.

params = struct;

% Objective function
params.cf = 'kl';   %  'is', 'kl', 'ed'; takes precedence over setting the beta value
  % alternately define: params.beta = 1;
params.sparsity = 5;

% Stopping criteria
params.max_iter = 100;
params.conv_eps = 1e-3;
% Display evolution of objective function
params.diplay   = 0;

% Random seed: any value over than 0 sets the seed to that value
params.random_seed = 1;

% Optional initial values for W
%params.init_w
% Number of components: if init_w is set and r larger than the number of
% basis functions in init_w, the extra columns are randomly generated
params.r = 500;
% Optional initial values for H: if not set, randomly generated
%params.init_h

% List of dimensions to update: if not set, update all dimensions.
%params.w_update_ind = true(r,1); % set to false(r,1) for supervised NMF
%params.h_update_ind = true(r,1);

[w, h, objective] = sparse_nmf(v, params);
```

B.2 sparse_nmf.m

```
function [w, h, objective] = sparse_nmf(v, params)

% SPARSE_NMF Sparse NMF with beta-divergence reconstruction error,
% L1 sparsity constraint, optimization in normalized basis vector space.
%
% [w, h, objective] = sparse_nmf(v, params)
%
% Inputs:
% v:  matrix to be factorized
% params: optional parameters
%     beta:     beta-divergence parameter (default: 1, i.e., KL-divergence)
%     cf:       cost function type (default: 'kl'; overrides beta setting)
%               'is': Itakura-Saito divergence
%               'kl': Kullback-Leibler divergence
%               'kl': Euclidean distance
%     sparsity: weight for the L1 sparsity penalty (default: 0)
%     max_iter: maximum number of iterations (default: 100)
%     conv_eps: threshold for early stopping (default: 0,
%                                             i.e., no early stopping)
%     display:  display evolution of objective function (default: 0)
%     random_seed: set the random seed to the given value
%                  (default: 1; if equal to 0, seed is not set)
%     init_w:   initial setting for W (default: random;
%                                      either init_w or r have to be set)
%     r:        # basis functions (default: based on init_w's size;
%                                 either init_w or r have to be set)
%     init_h:   initial setting for H (default: random)
%     w_update_ind: set of dimensions to be updated (default: all)
%     h_update_ind: set of dimensions to be updated (default: all)
%
% Outputs:
% w: matrix of basis functions
% h: matrix of activations
% objective: objective function values throughout the iterations
%
%
%
% References:
% J. Eggert and E. Korner, "Sparse coding and NMF," 2004
% P. D. O'Grady and B. A. Pearlmutter, "Discovering Speech Phones
%   Using Convolutive Non-negative Matrix Factorisation
%   with a Sparseness Constraint," 2008
% J. Le Roux, J. R. Hershey, F. Weninger, "Sparse NMF - half-baked or well
%   done?," 2015
%
% This implementation follows the derivations in:
% J. Le Roux, J. R. Hershey, F. Weninger,
% "Sparse NMF - half-baked or well done?,"
% MERL Technical Report, TR2015-023, March 2015
%
% If you use this code, please cite:
% J. Le Roux, J. R. Hershey, F. Weninger,
% "Sparse NMF - half-baked or well done?,"
% MERL Technical Report, TR2015-023, March 2015
%   @TechRep{LeRoux2015mar,
%     author = {{Le Roux}, J. and Hershey, J. R. and Weninger, F.},
%     title = {Sparse {NMF} - half-baked or well done?},
%     institution = {Mitsubishi Electric Research Labs (MERL)},
%     number = {TR2015-023},
%     address = {Cambridge, MA, USA},
%     month = mar,
%     year = 2015
%   }
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Copyright (C) 2015 Mitsubishi Electric Research Labs (Jonathan Le Roux,
%                                                         Felix Weninger, John R. Hershey)
%   Apache 2.0  (http://www.apache.org/licenses/LICENSE-2.0)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

m = size(v, 1);
n = size(v, 2);

if ~exist('params', 'var')
    params = struct;
end

```

```
78  if ~isfield(params, 'max_iter')
79      params.max_iter = 100;
80  end
81
82  if ~isfield(params, 'random_seed')
83      params.random_seed = 1;
84  end
85
86  if ~isfield(params, 'sparsity')
87      params.sparsity = 0;
88  end
89
90  if ~isfield(params, 'conv_eps')
91      params.conv_eps = 0;
92  end
93
94  if ~isfield(params, 'cf')
95      params.cf = 'kl';
96  end
97
98  switch params.cf
99      case 'is'
100         params.beta = 0;
101     case 'kl'
102         params.beta = 1;
103     case 'ed'
104         params.beta = 2;
105     otherwise
106         if ~isfield(params, 'beta')
107             params.beta = 1;
108         end
109 end
110
111 if params.random_seed > 0
112     rand('seed', params.random_seed);
113 end
114
115 if ~isfield(params, 'init_w')
116     if ~isfield(params, 'r')
117         error('Number of components or initialization must be given')
118     end
119     r = params.r;
120     w = rand(m, r);
121 else
122     ri = size(params.init_w, 2);
123     w(:, 1:ri) = params.init_w;
124     if isfield(params, 'r') && ri < params.r
125         w(:, (ri + 1) : params.r) = rand(m, params.r - ri);
126         r = params.r;
127     else
128         r = ri;
129     end
130 end
131
132 if ~isfield(params, 'init_h')
133     h = rand(r, n);
134 elseif ischar(params.init_h) && strcmp(params.init_h, 'ones')
135     fprintf('sup_nmf: Initalizing H with ones.\n');
136     h = ones(r, n);
137 else
138     h = params.init_h;
139 end
140
141 if ~isfield(params, 'w_update_ind')
142     params.w_update_ind = true(r, 1);
143 end
144
145 if ~isfield(params, 'h_update_ind')
146     params.h_update_ind = true(r, 1);
147 end
148
149 % sparsity per matrix entry
150 if length(params.sparsity) == 1
151     params.sparsity = ones(r, n) * params.sparsity;
152 elseif size(params.sparsity, 2) == 1
153     params.sparsity = repmat(params.sparsity, 1, n);
154 end
155
156 % Normalize the columns of W and rescale H accordingly
```

```
wn = sqrt(sum(w.^2));
w  = bsxfun(@rdivide,w,wn);
h  = bsxfun(@times,  h,wn');

if ~isfield(params, 'display')
    params.display = 0;
end

flr = 1e-9;
lambda = max(w * h, flr);
last_cost = Inf;

objective = struct;
objective.div = zeros(1,params.max_iter);
objective.cost = zeros(1,params.max_iter);

div_beta  = params.beta;
h_ind = params.h_update_ind;
w_ind = params.w_update_ind;
update_h = sum(h_ind);
update_w = sum(w_ind);

fprintf(1,'Performing sparse NMF with beta-divergence, beta=%.1f\n',div_beta);

tic
for it = 1:params.max_iter

    % H updates
    if update_h > 0
        switch div_beta
            case 1
                dph = bsxfun(@plus, sum(w(:, h_ind))', params.sparsity);
                dph = max(dph, flr);
                dmh = w(:, h_ind)' * (v ./ lambda);
                h(h_ind, :) = bsxfun(@rdivide, h(h_ind, :) .* dmh, dph);
            case 2
                dph = w(:, h_ind)' * lambda + params.sparsity;
                dph = max(dph, flr);
                dmh = w(:, h_ind)' * v;
                h(h_ind, :) = h(h_ind, :) .* dmh ./ dph;
            otherwise
                dph = w(:, h_ind)' * lambda.^(div_beta - 1) + params.sparsity;
                dph = max(dph, flr);
                dmh = w(:, h_ind)' * (v .* lambda.^(div_beta - 2));
                h(h_ind, :) = h(h_ind, :) .* dmh ./ dph;
        end
        lambda = max(w * h, flr);
    end


    % W updates
    if update_w > 0
        switch div_beta
            case 1
                dpw = bsxfun(@plus,sum(h(w_ind, :), 2)', ...
                    bsxfun(@times, ...
                    sum((v ./ lambda) * h(w_ind, :)' .* w(:, w_ind)), w(:, w_ind)));
                dpw = max(dpw, flr);
                dmw = v ./ lambda * h(w_ind, :)' ...
                    + bsxfun(@times, ...
                    sum(bsxfun(@times, sum(h(w_ind, :),2)', w(:, w_ind))), w(:, w_ind));
                w(:, w_ind) = w(:,w_ind) .* dmw ./ dpw;
            case 2
                dpw = lambda * h(w_ind, :)' ...
                    + bsxfun(@times, sum(v * h(w_ind, :)' .* w(:, w_ind)), w(:, w_ind));
                dpw = max(dpw, flr);
                dmw = v * h(w_ind, :)' + ...
                    bsxfun(@times, sum(lambda * h(w_ind, :)' .* w(:, w_ind)), w(:, w_ind));
                w(:, w_ind) = w(:,w_ind) .* dmw ./ dpw;
            otherwise
                dpw = lambda.^(div_beta - 1) * h(w_ind, :)' ...
                    + bsxfun(@times, ...
                    sum((v .* lambda.^(div_beta - 2)) * h(w_ind, :)' .* w(:, w_ind)), ...
                    w(:, w_ind));
                dpw = max(dpw, flr);
                dmw = (v .* lambda.^(div_beta - 2)) * h(w_ind, :)' ...
                    + bsxfun(@times, ...
                    sum(lambda.^(div_beta - 1) * h(w_ind, :)' .* w(:, w_ind)), w(:, w_ind));
                w(:, w_ind) = w(:,w_ind) .* dmw ./ dpw;
```

```
        end
        % Normalize the columns of W
        w = bsxfun(@rdivide,w,sqrt(sum(w.^2)));
        lambda = max(w * h, flr);
    end


    % Compute the objective function
    switch div_beta
        case 1
            div = sum(sum(v .* log(v ./ lambda) - v + lambda));
        case 2
            div = sum(sum((v - lambda) .^ 2));
        case 0
            div = sum(sum(v ./ lambda - log ( v ./ lambda) - 1));
        otherwise
            div = sum(sum(v.^div_beta + (div_beta - 1)*lambda.^div_beta ...
                - div_beta * v .* lambda.^(div_beta - 1))) / (div_beta * (div_beta - 1));
    end
    cost = div + sum(sum(params.sparsity .* h));

    objective.div(it)  = div;
    objective.cost(it) = cost;

    if params.display ~= 0
        fprintf('iteration %d div = %.3e cost = %.3e\n', it, div, cost);
    end

    % Convergence check
    if it > 1 && params.conv_eps > 0
        e = abs(cost - last_cost) / last_cost;
        if (e < params.conv_eps)
            disp('Convergence reached, aborting iteration')
            objective.div = objective.div(1:it);
            objective.cost = objective.cost(1:it);
            break
        end
    end
    last_cost = cost;
end
toc

end
```

References

[1] J. Le Roux, "Exploiting regularities in natural acoustical scenes for monaural audio signal estimation, decomposition, restoration and modification," Ph.D. dissertation, The University of Tokyo & Université Paris VI – Pierre et Marie Curie, Mar. 2009.

[2] P. D. Ogrady and B. A. Pearlmutter, "Discovering speech phones using convolutive non-negative matrix factorisation with a sparseness constraint," *Neurocomputing*, vol. 72, no. 1, pp. 88–101, 2008.

[3] A. Cichocki, R. Zdunek, A. H. Phan, and S.-i. Amari, *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. John Wiley & Sons, 2009.

[4] C. Févotte and J. Idier, "Algorithms for nonnegative matrix factorization with the beta-divergence," *Neural Computation*, vol. 23, no. 9, pp. 2421–2456, 2011.

[5] P. Smaragdis, "Convolutive speech bases and their application to supervised speech separation," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, no. 1, pp. 1–14, 2007.

[6] C. Févotte, N. Bertin, and J.-L. Durrieu, "Nonnegative matrix factorization with the Itakura-Saito divergence: with application to music analysis," *Neural Computation*, vol. 21, no. 3, pp. 793–830, March 2009.

[7] W. Liu, N. Zheng, and X. Lu, "Non-negative matrix factorization for visual coding," in *Proc. ICASSP*, vol. 3, 2003.

[8] M. N. Schmidt and R. K. Olsson, "Single-channel speech separation using sparse non-negative matrix factorization," in *Proc. Interspeech*, 2006.

[9] J. Eggert and E. Körner, "Sparse coding and NMF," in *Proc. of Neural Networks*, vol. 4, Dalian, China, 2004, pp. 2529–2533.

[10] B. Raj, T. Virtanen, S. Chaudhuri, and R. Singh, "Non-negative matrix factorization based compensation of music for automatic speech recognition," in *Proc. of Interspeech*, Makuhari, Japan, 2010, pp. 717–720.

[11] J. T. Geiger, F. Weninger, A. Hurmalainen, J. F. Gemmeke, M. Wöllmer, B. Schuller, G. Rigoll, and T. Virtanen, "The TUM+TUT+KUL approach to the CHiME Challenge 2013: Multi-stream ASR exploiting BLSTM networks and sparse NMF," in *Proc. of 2nd CHiME Workshop held in conjunction with ICASSP 2013*. Vancouver, Canada: IEEE, 2013, pp. 25–30.

[12] P. Smaragdis, M. Shashanka, and B. Raj, "A sparse non-parametric approach for single channel separation of known sounds," in *Proc. NIPS*, 2009, pp. 1705–1713.

[13] B. King and L. Atlas, "Single-channel source separation using simplified-training complex matrix factorization," in *Proc. ICASSP*, 2010, pp. 4206–4209.

[14] H. Kameoka, N. Ono, K. Kashino, and S. Sagayama, "Complex NMF: A new sparse representation for acoustic signals," in *Proc. ICASSP*, 2009, pp. 3437–3440.

[15] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.

[16] M. Mørup, M. N. Schmidt, and L. K. Hansen, "Shift invariant sparse coding of image and music data," Technical University of Denmark, Tech. Rep. IMM2008-04659, 2008.

[17] S. C. Douglas, S.-I. Amari, and S.-Y. Kung, "Gradient adaptation with unit-norm constraints," Southern Methodist University, Tech. Rep. EE-99-003, 1999.

[18] T. P. Krasulina, "Method of stochastic approximation in the determination of the largest eigenvalue of the mathematical expectation of random matrices," *Automat. Remote Control*, vol. 2, pp. 215–221, 1970.

[19] V. U. Reddy, B. Egardt, and T. Kailath, "Least squares type algorithm for adaptive implementation of Pisarenko's harmonic retrieval method," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 30, no. 3, pp. 399–405, Jun. 1982.

[20] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Proc. of NIPS*, Vancouver, Canada, 2001, pp. 556–562.

[21] C. Joder, F. Weninger, F. Eyben, D. Virette, and B. Schuller, "Real-time speech separation by semi-supervised nonnegative matrix factorization," in *Proc. of LVA/ICA*, ser. Lecture Notes in Computer Science, F. J. Theis, A. Cichocki, A. Yeredor, and M. Zibulevsky, Eds., vol. 7191. Tel Aviv, Israel: Springer, March 2012, pp. 322–329, special Session Real-world constraints and opportunities in audio source separation.

[22] P. Smaragdis, B. Raj, and M. Shashanka, "Supervised and semi-supervised separation of sounds from single-channel mixtures," in *Proc. of ICA*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 414–421.

[23] E. Vincent, R. Gribonval, and C. Févotte, "Performance measurement in blind audio source separation," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 14, no. 4, pp. 1462–1469, Jul. 2006.