Optimization and Parallelization of Monaural Source Separation Algorithms in the openBliSSART Toolkit

Felix Weninger · Björn Schuller

Abstract We describe the implementation of monaural audio source separation algorithms in our toolkit openBliSSART (Blind Source Separation for Audio Recognition Tasks). To our knowledge, it provides the first freely available C++ implementation of Non-Negative Matrix Factorization (NMF) supporting the Compute Unified Device Architecture (CUDA) for fast parallel processing on graphics processing units (GPUs). Besides integrating parallel processing, openBliSSART introduces several numerical optimizations of commonly used monaural source separation algorithms that reduce both computation time and memory usage. By illustrating a variety of use-cases from audio effects in music processing to speech enhancement and feature extraction, we demonstrate the wide applicability of our application framework for a multiplicity of research and end-user applications. We evaluate the toolkit by benchmark results of the NMF algorithms and discuss the influence of their parameterization on source separation quality and real-time factor. In the result, the GPU parallelization in openBliSSART introduces double-digit speedups with respect to conventional CPU computation, enabling real-time processing on a desktop PC even for high matrix dimensions.

This work was supported by the Federal Republic of Germany through the German Research Foundation (DFG) under grant no. SCHU 2508/2-1 ("Non-Negative Matrix Factorization for Robust Feature Extraction in Speech Processing").

F. Weninger B. Schuller
Institute for Human-Machine Communication,
Technische Universität München,
80290 München, Germany
e-mail: weninger@tum.de

1 Introduction

1.1 Background

Blind Audio Source Separation (BASS) is a challenging, yet promising field in signal processing. Its applications are manifold; BASS can be used for auditory scene analysis [10] or as an intelligent filter for Music Information Retrieval (MIR) tasks, music remixing and automatic speech recognition (ASR). The last years have seen a growing number of approaches exploiting Non-Negative Matrix Factorization (NMF) [2], whose most prominent advantage is that it can extract an arbitrary number of sources from monophonic signals, in contrast to independent component analysis [13] or similar microphone array methods. A typical application of NMF in audio editing is the retrieval of single tracks from professionally recorded (mono- or stereophonic) music, such as drums [11, 21] or the leading voice [4] ('karaoke application'). In MIR, NMF can be used for polyphonic transcription ('waveform to MIDI converter') [25]; furthermore, extraction of the singer's voice can improve the recognition of lyrics, which appears to be a particularly challenging task [15]. In the field of speech processing, monaural BASS techniques based on NMF can deliver enhanced robustness of ASR by separating the wanted speech from interfering signals such as background noise [31, 36], music [19] or even other speakers [3, 16, 24]. Beyond its capability to extract sources from audio signals, NMF can be exploited for audio feature extraction [22, 34], especially in highly noisy conditions [7, 23, 33] due to its inherent separation capabilities.

With the increasing amount of computational power available today, especially the parallel floating point computation capabilities of industry standard graphics processing units (GPUs), we are moving towards the point where NMF-based algorithms are ready to be integrated in real-time capable end-user applications [8]. Still, the lion's share of the studies in the field focuses on optimizing the separation results, neglecting implementation issues. Besides, few open-source implementations of general-purpose, NMF-based source separation algorithms exist, with the notable exception of the FASST source separation toolbox [18]. FASST provides a flexible BASS framework implemented in Matlab; its focus is on automatic exploitation of prior knowledge about the sources, as specified by the user. The openBliSSART toolkit, however, emphasizes performance and integrability into existing end-user applications by providing a modular C++ framework. By separating core algorithms from pre- and post-processing components, it can be seamlessly integrated into speech and music processing applications. Source code and demonstrations can be found at http://openblissart.github.com/openBliSSART. We have introduced openBliSSART in [32]; since then, one remarkable development has been to parallelize computationally intensive parts of the algorithms on GPUs following the Compute Unified Device Architecture (CUDA) standard. An earlier study [1] proposed the usage of CUDA for NMF, but its evaluation was limited to a single NMF algorithm and the matrix parameterization typically encountered in musical instrument separation. Similarly, in [8], the use case of exemplar-based speech classification by parallel NMF was considered exclusively, which corresponds to a very high dimensional factorization. In contrast, one of the main contributions of this article is a large-scale performance evaluation of parallelized NMF, oriented on a variety of use cases from speech and music source separation and audio feature extraction, which typically result in the usage of different NMF algorithms and greatly varying input dimensions.

In the remainder of this article, we will first describe the algorithmic framework implemented in openBliSSART in Section 2. From a description of the generic algorithms, we will derive numerical optimizations for special cases relevant in practice, including novel optimizations for non-negative matrix deconvolution (Section 3). We address the impact of floating point precision and usage of GPUs on the real-time capability of the system and conclude with benchmark performances in application scenarios from speech processing in Section 4.

1.2 Notations

To increase clarity of the following sections, we introduce the following notations: For a matrix $\mathbf{A}$, $\mathbf{A}_{i,j}$ denotes the element at row i and column j, and the notation $\mathbf{A}_{i,:}$—resembling Matlab syntax—denotes the i-th row of $\mathbf{A}$ (as a row vector); we analogously define $\mathbf{A}_{:,j}$ for the j-th column of $\mathbf{A}$ (as a column vector). We write $\mathbf{A} \otimes \mathbf{B}$ for the element-wise product of matrices $\mathbf{A}$ and $\mathbf{B}$; division of matrices is always to be understood as element-wise. We write $[\mathbf{A}, \mathbf{B}]$ for the column-wise concatenation of matrices as in Matlab syntax. $\mathbf{0}^{M\times N}$ and $\mathbf{1}^{M\times N}$ denote all-zero and all-one matrices of dimension $M \times N$. Finally, for a matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$ and $p \geq 0$, we define

$$\overset{p\rightarrow}{\mathbf{A}} := \begin{cases} \left[\mathbf{0}^{M\times p}\,,\ \mathbf{A}_{:,1:N-p}\right] & p > 0 \\ \mathbf{A} & p = 0 \end{cases}, \qquad (1)$$

i.e., the entries of $\mathbf{A}$ are shifted p spots to the right, filling with zeros from the left. Analogously, with $\overset{\leftarrow p}{\mathbf{A}}$ we denote a 'left-shift' version of $\mathbf{A}$, where zeros are introduced from the right.

2 Algorithmic Framework: Signal Separation and Reconstruction

2.1 Component Separation Algorithms

openBliSSART's design is oriented on BASS techniques realized by NMF. A basic procedure is to extract an arbitrary number R of sources (*components*) from audio files by computing the non-negative factorization of a spectrogram matrix $\mathbf{V} \in \mathbb{R}_+^{M\times N}$ obtained from short-time Fourier transformation (STFT) into a spectral basis $\mathbf{W} \in \mathbb{R}_+^{M\times R}$ and activation matrix $\mathbf{H} \in \mathbb{R}_+^{R\times N}$:

$$\mathbf{V} \approx \mathbf{W}\mathbf{H}, \qquad (2)$$

yielding R component spectrograms $\mathbf{V}^{(j)}$, $j = 1, \ldots, R$ either by multiplication of each basis vector $\mathbf{w}^{(j)} := \mathbf{W}_{:,j}$ with its activations $\mathbf{h}^{(j)} := \mathbf{H}_{j,:}$, as in [29], or by a 'Wiener filter' approach as described in [5, 24]:

$$\mathbf{V}^{(j)} = \mathbf{V} \otimes \frac{\mathbf{w}^{(j)}\mathbf{h}^{(j)}}{\mathbf{W}\mathbf{H}}. \qquad (3)$$

The latter ensures that $\sum_j \mathbf{V}^{(j)} = \mathbf{V}$; hence, no information is lost due to the factorization. Each $\mathbf{V}^{(j)}$ is then transformed back to the time domain by inverse STFT and overlap-add.

To obtain a factorization according to Eq. 2, a variety of NMF algorithms can be used that minimize a distance function $d(\mathbf{V}|\mathbf{WH})$ by multiplicative updates of the matrices, starting from a random initialization. $d(\mathbf{V}|\mathbf{WH})$ can be chosen as the β-divergence or one of its special instances, the Itakura-Saito (IS) [5] divergence, Kullback–Leibler (KL) divergence, or squared Euclidean distance (ED) [14]. The choice of β depends on the statistical properties of the signal; hence, openBliSSART implements a generic algorithm for the β-divergence as well as optimizations for special cases, as laid out in Section 3.3. Besides, to support overcomplete decomposition, i.e., choosing R such that $R(M+N) \gg MN$, sparse NMF variants [29] for any of the aforementioned distance functions, as well as the sparse Euclidean NMF variant used in [20], are implemented.

In addition to basic NMF, non-negative matrix deconvolution (NMD) [24, 30] is provided as a context-sensitive NMF extension where each component is characterized by a sequence of spectra, rather than by an instantaneous observation. Hence, NMD is a generalization of Non-Negative Matrix Factorization (NMF). In an NMF-alike notation, the signal model underlying NMD can be written as follows, denoting the approximation of $\mathbf{V}$ by $\mathbf{\Lambda}$:

$$\mathbf{V} \approx \sum_{p=0}^{P-1} \mathbf{W}(p) \overset{p\rightarrow}{\mathbf{H}} =: \mathbf{\Lambda}, \tag{4}$$

where $\rightarrow$ is the 'shift' operator defined in Eq. 1. To facilitate discussion in the subsequent sections, the term 'NMF' will refer to NMF or NMD unless explicitly stated otherwise.

The aforementioned NMF algorithms can be run on magnitude, power, and Mel-scale spectra. As an additional transformation of the spectrogram $\mathbf{V}$, a sliding window can be applied as in [7], transforming the original spectrogram $\mathbf{V}$ to a matrix $\mathbf{V}'$ whose columns correspond to overlapping sequences of short-time spectra in $\mathbf{V}$. This provides a contextual factorization as does NMD, but with each sliding window factorized separately; whether this approach is superior to NMD seems to depend on the application [12]. Note that openBliSSART also implements inverse operations to the aforementioned transformations of the spectrogram—including Mel filtering and sliding window—to allow proper signal reconstruction.

2.2 Supervised Component Classification

To cope with scenarios such as instrument separation—as in [11, 21]—it is necessary to extend the basic source separation capabilities: Here, typically 20–40 NMF components are needed for appropriate signal modeling, thus the 'tracks' corresponding to one instrument, or an instrument class such as drums, generally comprise more than one NMF component. Consequently, a classification process is necessary to overlay individual components into C class spectrograms, yielding the procedure depicted in Fig. 1: First, a selection of training signals is separated by means of NMF. Subsequently, the resulting components are annotated (e.g., as drum or harmonic sounds), and features are extracted from them to train a classifier, e.g., a Support Vector Machine (SVM). Then, the actual separation process performs NMF and uses the previously trained classifier on the features of the separated components to overlay them into class spectrograms $\mathbf{V}_c$, $c = 1, \ldots, C$: Defining

$$J_c = \{j : (\mathbf{w}^{(j)}, \mathbf{h}^{(j)}) \text{ classified as class } c\},$$
$$\mathbf{V}_c = \sum_{j \in J_c} \mathbf{V}^{(j)}. \tag{5}$$

For convenient annotation of the training components, openBliSSART provides a graphical user interface. Acoustic features which are used for classification include Mel-frequency Cepstral Coefficients or features specially suited to instrument separation, such as noise-likeness or periodicity of the time-varying gains [27]. The available NMF algorithms are shown in Table 1, yielding a flexible framework for NMF-based source separation where preprocessing, factorization, and component reconstruction algorithms can be chosen independently.

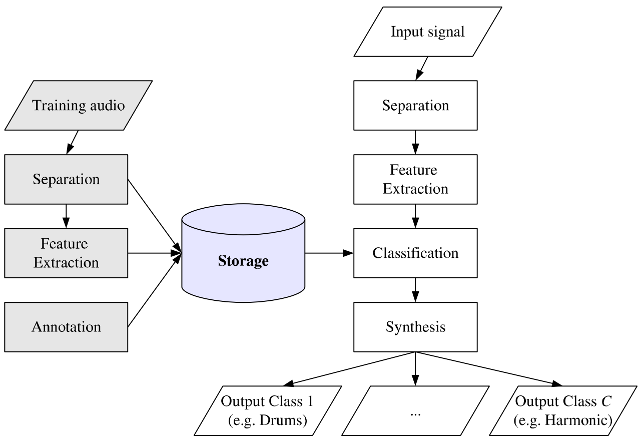


Figure 1 Supervised component classification, as in instrument separation: the openBliSSART storage module manages the components from which a classifier is built for usage in the separation process. The steps required to train the classifier are depicted in *gray shade*.

Table 1 Spectrogram preprocessing, factorization (according to a cost function $d(\mathbf{V}|\mathbf{WH})$), and signal reconstruction algorithms for monaural source separation implemented in openBliSSART.

Preprocessing	Algorithms	$d(\mathbf{V}\|\mathbf{WH})$	Reconstruction
Mel filter	NMF	IS div.	Default
Power spec.	NMD	KL div.	Wiener filtering
Sliding window		Eucl. dist. + sparsity	Comp. classif.

2.3 (Semi-)Supervised NMF and Acoustic Feature Extraction

As an alternative method to integrate a-priori knowledge about the sources into NMF, *supervised NMF* is provided in openBliSSART (cf. Fig. 2). Here, the first NMF factor ($\mathbf{W}$) is pre-defined as a set of spectra that are characteristic for the sources to be separated, as opposed to a random initialization, and only the second NMF factor is computed through multiplicative updates. A typical application is speech enhancement, where the sources comprise different persons speaking simultaneously [20, 24], or speech and noise [24, 31]. The spectra used for initialization can be estimated by NMF decomposition of training material, as in [20, 24, 31]. An alternative is to directly use training segments [19]; openBliSSART supports this by allowing initialization with the spectrograms of training samples. Using any of the methods shown in Table 1, time signals corresponding to different sources (e.g., speakers) can be synthesized by adding up component spectrograms. In supervised NMF, no classification step is needed, since the assignment of components to sources is defined a priori.

Besides its usage for source separation, openBliSSART was the toolkit used in our research on supervised NMF feature extraction, which has delivered excellent results in robust speech processing [7] including detection of non-linguistic vocalizations and emotion [33, 34]. Thereby different sets of components are selected as an in- or overcomplete feature basis, for which the time-varying activation matrix (i.e., the second NMF factor) is computed by supervised NMF. Similarly to supervised NMF for source separation, this basis can be estimated by means of NMF from training spectrograms—as, e. g., in [22, 23]—or, these can be used directly as a basis. In both cases, training data is assumed to be labeled, e. g., with words [7] or sound classes [23, 34], and thus the activations of the basis vectors directly reveal the content of the audio signal [7, 23]. openBliSSART allows this matrix to be exported for further processing, e.g., by popular research toolkits such as Weka [9] or HTK (Hidden Markov Model Toolkit) [38], enabling the usage of NMF features for a huge variety of research.

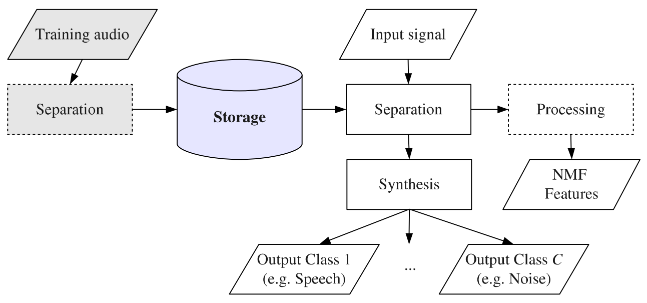


Figure 2 Supervised NMF: a set of spectral components (which can themselves be computed by NMF from training audio) serve as constant basis for NMF; the activations can be exported as features or be used to synthesize audio signals for the sources without the classification step from Fig. 1.

Finally, openBliSSART also supports semi-supervised NMF [17, 23, 26]: In that case, only part of the $\mathbf{W}$ matrix is pre-defined and kept constant throughout the NMF iterations; this allows unknown, variable sources such as instationary additive noise to be estimated 'on-the-fly' during source separation or NMF feature extraction. The 'free' NMF components in semi-supervised NMF can be input to component classification as in the previous section.

3 Numerical Optimizations

To describe the numerical optimizations that openBliSSART provides for NMD and NMF, we first recall the multiplicative update rules at the core of the source separation algorithms, which make up for the lion's share of the computational effort. In turn, they are dominated by dense matrix-matrix multiplication, which can be parallelized very efficiently by 'off-the-shelf' libraries, especially those provided by GPU vendors. Other operations such as element-wise matrix operations do require some 'hand-crafted' routines (cf. Section 3.4), but are less critical for performance [1]. The update rules for the matrices $\mathbf{W}(p)$, $p = 0, \ldots, P-1$ as well as $\mathbf{H}$ in NMD were given for the KL divergence cost function in [24] and for the Euclidean distance in [30]. In [16], they were generalized to the general beta-divergence. The formulation of the $\mathbf{W}(p)$ update rule using matrix products is as follows:

$$\mathbf{W}(p) \leftarrow \mathbf{W}(p) \otimes \frac{(\mathbf{V} \otimes \mathbf{\Lambda}^{\beta-2})(\overset{p\rightarrow}{\mathbf{H}})^T}{\mathbf{\Lambda}^{\beta-1}(\overset{p\rightarrow}{\mathbf{H}})^T}. \qquad (6)$$

The rule 6 is applied for $p = 0, \ldots, P-1$. Note that in Eq. 6, the shifted matrix $\mathbf{H}$ is transposed, not vice versa. After updating each $\mathbf{W}(p)$, the value of $\mathbf{\Lambda}$ has to be recomputed; however, doing so by using Eq. 4 takes $O(P^2)$ matrix products per iteration, which can be decreased to $O(P)$ by using the difference-based formulation from [30]. The latter is implemented in openBliSSART.

The update rule for $\mathbf{H}$ implemented in openBliSSART is a slightly modified version of the rule proposed in [24, 30], which averages over the influence that every $\mathbf{W}(p)$ has on $\mathbf{H}$. Our formulation avoids the divisions by zero in the rightmost columns of the right hand side operands, which occur in the original formulation of the algorithms:

$$\mathbf{H}_{j,t} \leftarrow \mathbf{H}_{j,t} \frac{1}{P(t)} \sum_{p=0}^{P(t)-1} \left[\frac{\mathbf{W}(p)^T (\mathbf{V} \otimes \overset{\leftarrow p}{\mathbf{\Lambda}^{\beta-2}})}{\mathbf{W}(p)^T \overset{\leftarrow p}{\mathbf{\Lambda}^{\beta-1}}} \right]_{j,t} , \quad (7)$$

with $j = 1, \ldots, R$, $t = 1, \ldots, N$ and $P(t) := \min\{P, N-t+1\}$.

3.1 Elimination of Shift Operators in NMD

Overall, the above update rules in 'matrix form' are compact, and can be implemented very efficiently using linear algebra routines employing vectorization (cf. Section 3.4). Still, if implemented straightforwardly, the shift operators introduce additional operations and increase memory usage. Hence, we will show how to eliminate them completely by reducing multiplications with shifted matrices to multiplication of submatrices. This is in contrast to [2] where usage of special Matlab functions has been proposed to eliminate the shifts. Observe that the shift operators are always used within a matrix-matrix multiplication, introducing zeros into one of the factors: Thus, the shifting can be 'simulated' by adjusting the summation ranges in the scalar products used in matrix multiplication. This allows an easy and very efficient implementation without having to compute (and store) shifted versions of the matrices, or submatrices: The BLAS (Basic Linear Algebra Subroutines) standard supports submatrix multiplications directly on the memory blocks corresponding to the full matrices. Defining $\tilde{\mathbf{V}} := \mathbf{V} \otimes \mathbf{\Lambda}^{\beta-2}$, the numerator of the rule 6 can be transformed using

$$\begin{aligned} \tilde{\mathbf{V}}(\overset{p\rightarrow}{\mathbf{H}})^T &= \tilde{\mathbf{V}} \left[\mathbf{0}^{R\times p} , \mathbf{H}_{:,1:N-p} \right]^T \\ &= \tilde{\mathbf{V}}_{:,p+1:N} \mathbf{H}^T_{:,1:N-p} \quad , \end{aligned} \quad (8)$$

and the numerator of the rule 7 can be reformulated using

$$\begin{aligned} \mathbf{W}(p)^T \overset{\leftarrow p}{\tilde{\mathbf{V}}} &= \mathbf{W}(p)^T \left[\tilde{\mathbf{V}}_{:,p+1:N} , \mathbf{0}^{M\times p} \right] \\ &= \left[\left(\mathbf{W}(p)^T \tilde{\mathbf{V}}_{:,p+1:N} \right) , \mathbf{0}^{M\times p} \right] \quad . \end{aligned} \quad (9)$$

The denominators of the rules can be transformed accordingly. Finally, the shifts in the computation of $\mathbf{\Lambda}$ (Eq. 4) can be eliminated by exploiting the equality

$$\begin{aligned} \mathbf{W}(p) \overset{p\rightarrow}{\mathbf{H}} &= \mathbf{W}(p) \left[\mathbf{0}^{R\times p} , \mathbf{H}_{:,1:N-p} \right] \\ &= \left[\mathbf{0}^{M\times p} , \left(\mathbf{W}(p) \mathbf{H}_{:,1:N-p} \right) \right] \quad , \end{aligned} \quad (10)$$

thereby eliminating all shift operators from the algorithm.

3.2 Optimization of Kullback–Leibler NMD

For the KL divergence ($\beta = 1$), one can eliminate the matrix multiplication from the denominator of Eq. 6 by computing row sums for $j = 1, \ldots, R$:

$$\left(\mathbf{1}^{M\times N} \left(\overset{p\rightarrow}{\mathbf{H}} \right)^T \right)_{i,j} = \sum_{t=1}^{N-p} \mathbf{H}_{j,t} \quad , \quad (11)$$

reducing the effort to $O(RN)$ operations. Analogously, one 'reduces' the denominator of Eq. 7 to a column sum for $t = 1, \ldots, N$:

$$\left(\mathbf{W}(p)^T \overset{\leftarrow p}{\mathbf{1}^{M\times N}} \right)_{j,t} = \begin{cases} \sum_{i=1}^{M} \mathbf{W}(p)_{i,j} & t \le N-p \\ 0 & t > N-p. \end{cases} \quad (12)$$

Since calculation of these row and column sums consumes only a fraction of the total processing time [1], we simply implement them by a scalar product with an all-one vector. Note that due to our formulation of the $\mathbf{H}$ update rule (Eq. 7), no division by zero occurs in practice. Furthermore, we point out that in contrast to the algorithm implemented in openBliSSART (Eqs. 11 and 12), the original formulation of the KL divergence NMD algorithm [24] ignored the fact that the $\mathbf{1}$ matrix in the update rule for $\mathbf{H}$ must be shifted. As a result, the original implementation sums up zero updates in the average update for $\mathbf{H}$, which makes the rightmost P columns of $\mathbf{H}$ effectively vanish after a few iterations, depriving the algorithm of RP degrees of freedom and leading to slower convergence especially for short signals.

3.3 Optimization of Euclidean NMF

In case of NMD minimizing Euclidean distance ($\beta = 2$), the update rules (Eqs. 6 and 7) are greatly simplified, as the element-wise multiplication with $\mathbf{\Lambda}^{\beta-2} = \mathbf{1}$ is eliminated. Besides, in case of Euclidean NMF ($P = 1$), the update rules are further reduced to the ones originally proposed in [14]. In matrix formulation, they read

$$\mathbf{H} \leftarrow \mathbf{H} \otimes \frac{\mathbf{W}^T\mathbf{V}}{\mathbf{W}^T\mathbf{W}\mathbf{H}} \quad \text{and} \tag{13}$$

$$\mathbf{W} \leftarrow \mathbf{W} \otimes \frac{\mathbf{V}\mathbf{H}^T}{\mathbf{W}\mathbf{H}\mathbf{H}^T}. \tag{14}$$

In these rules, we can rearrange the matrix products by using their associativity. First, we consider the denominator of the $\mathbf{H}$ update rule 13, which contains the product $\mathbf{W}^T\mathbf{W}\mathbf{H}$. When executed in the order $\mathbf{W}^T(\mathbf{W}\mathbf{H})$, the computational complexity is $O(MNR)$; in contrast, it is $O(R^2(M+N))$ for the order $(\mathbf{W}^T\mathbf{W})\mathbf{H}$, assuming the standard matrix multiplication algorithm. Thus, the effort for the first case is expected to be lower if and only if $MN < R(M+N)$, that is, in case of overcomplete factorization. The nested matrix product $\mathbf{W}\mathbf{H}\mathbf{H}^T$ in the denominator of rule 14 can be treated analogously. Naturally, these are only asymptotic considerations, which are however supported by our experimental results using efficient linear algebra routines (Section 4). Hence, the openBliSSART routine for ED-NMF uses the computation order $(\mathbf{W}\mathbf{H})\mathbf{H}^T$ and $\mathbf{W}^T(\mathbf{W}\mathbf{H})$ for $MN < R(M+N)$, and $\mathbf{W}(\mathbf{H}\mathbf{H}^T)$ and $(\mathbf{W}^T\mathbf{W})\mathbf{H}$ otherwise. For evaluation of this strategy in the next section, the algorithms that always use the former or latter computation order will subsequently be denoted by 'NMF-ED-ov' and 'NMF-ED-in', as they are arguably optimized to either over- or incomplete factorization.

3.4 Implementation

For a practical implementation of the NMF algorithms using the above considerations, high performance linear algebra routines are needed. For CPU computation, we opt for the open-source BLAS implementation provided by the ATLAS project [35]. The ATLAS libraries can be considered industry standard and are at the core of Matlab's linear algebra capabilities. From our experience, the ATLAS routines decrease the real-time-factor (RTF) by an order of magnitude for typical NMF applications, compared to a 'naïve' matrix multiplication routine implemented in C++. Similarly, for GPU computation, we rely on the 'CUBLAS' implementation of the BLAS standard in NVIDIA's CUDA Computing Software Development Kit (SDK). However, some operations needed for NMF are not foreseen as routines in the BLAS standard, such as in the update of the $\mathbf{H}$ matrix performed by the KL-NMF algorithm, which (conceptionally) features an element-wise division of the $\mathbf{H}$ matrix by a vector of column sums (cf. Eqs. 7 and 12). In order to run the NMF algorithm entirely on the GPU without time intensive memory transfers to the CPU memory, openBliSSART extends CUBLAS by specialized CUDA 'kernels' implementing such operations. Kernels are routines that are automatically distributed to the GPU cores for vector processing in a 'single instruction, multiple data' (SIMD) fashion. For example, in the element-wise division mentioned above, every thread is assigned a submatrix of $\mathbf{H}$, and a pointer to the corresponding column sums, which can be processed independently without expensive inter-thread communication. More details on the implementation of CUDA kernels can be found in [1].

For further performance enhancement, openBliSSART utilizes FFTW (Fastest Fourier Transform in the West) [6] to realize Fast Fourier Transformation (FFT) for arbitrary (even prime) vector sizes, disposing of the need for zero padding. FFTW is particularly well suited to short-time Fourier transformation, as the library automatically pre-computes the best algorithmic strategy for a given Discrete Fourier Transform (DFT) size through on-the-fly performance measurements, then uses that strategy for all subsequent calculations. As the complexity of STFT is linear in the length of the audio signal—precisely, the number of frames—, and each frame can be processed independently, we expect it to be fruitful to integrate a parallel GPU version of the FFT algorithm into openBliSSART in the future.

4 Experimental Evaluation

In this section, we provide benchmark results achieved with the openBliSSART toolkit. We start first with an evaluation of the numerical optimization of Euclidean distance NMF in practice. We then move on to demonstrate the variety of means that openBliSSART provides to adjust the trade-off between computation time and separation quality, including parallelization, single- or optional double-precision calculation on both CPU and GPU, and parameterization of NMF itself, including various cost functions, the number of iterations, and the spectrogram dimension (DFT window size and frame shift).

All computation was performed on a desktop PC running Ubuntu Linux 10.04. The PC had an Intel Core2Quad CPU with 2.4 GHz clock frequency and 4 GB of RAM, and an NVIDIA GeForce GTX560 GPU with 336 CUDA cores—each with 810 MHz core and 160 MHz shader frequency—and 2 GB of RAM.

CPU computation was performed using a single computation thread (i. e., using only one of the four cores), in order to minimize singular effects due to interferences with concurrent system processes, to reflect an end-user application where only part of the CPU computation power can be dedicated to audio processing, and to compare to a baseline with no parallelization across computation units. However, intra-core vectorized processing on the CPU is performed by the employed ATLAS library, exploiting the Intel Multimedia Extensions (MMX) and their follow-up technologies.

4.1 Numerical Optimization

In Fig. 3, the processing times for in- and overcomplete factorization by NMF minimizing the ED function are shown, for CPU computation. For the 'component-level' evaluation of NMF in this experiment, we used random matrices, as both algorithms are numerically equivalent and separation quality is thus unaffected by the choice of algorithm. By comparing the processing times for the 'openBliSSART' strategy to either of the algorithms optimized to in- or overcomplete factorization ('NMF-ED-in', 'NMF-ED-ov') according to Section 3.3, it can be seen that the proposed implementation that determines the optimal algorithm by the factorization dimensionality leads to considerable decrease of computational effort, especially for very low or very high numbers of components.

4.2 Parallelization on Graphics Processing Units

Next, we evaluate the influence of computation architecture (CPU vs. GPU) on the real-time factors, using matrix dimensions encountered in typical applications in speech and music processing. It is of high interest to assess whether there is a 'break-even point' for GPU calculation: It could be expected that due to the overhead introduced by CPU-to-GPU memory transfers and thread synchronization of parallel computations on the GPU, GPU can outperform CPU calculation only for 'sufficiently high' matrix dimensions. Furthermore, we evaluate the impact of floating point precision for both CPU and GPU, as current generation GPUs increasingly support double precision calculations—they are increasingly popular in scientific computing applications, where the required precision depends on the application [37].

In Fig. 4, we show the processing time of the NMF algorithm (KL divergence) on a 500 × 1,000 matrix. This corresponds to a 1,001-point DFT (≈63 ms window size at 16 kHz sampling rate) and 10 s signal length at 10 ms frame shift (as used in speech feature extraction by NMF in [7, 23]) or, longer signals analyzed with larger frame shifts. The number of NMF components was varied from 5 to 5,000 to reflect NMF applications reaching from the extraction from a few number of simple sources [10] to high-dimensional decomposition [7]. Comparing single vs. double precision, we observe a consistent speedup from 1.9 up to 3.5 for GPU computation, depending on R. In contrast, for CPU computation, the results are more mixed when using lower numbers of components; particularly, for $R = 50$, double precision is about 1.2 times faster than single precision. Furthermore, in case of single precision, the measured RTF is not linear in the number of components R. We attribute these phenomena to peculiarities of the employed ATLAS library.

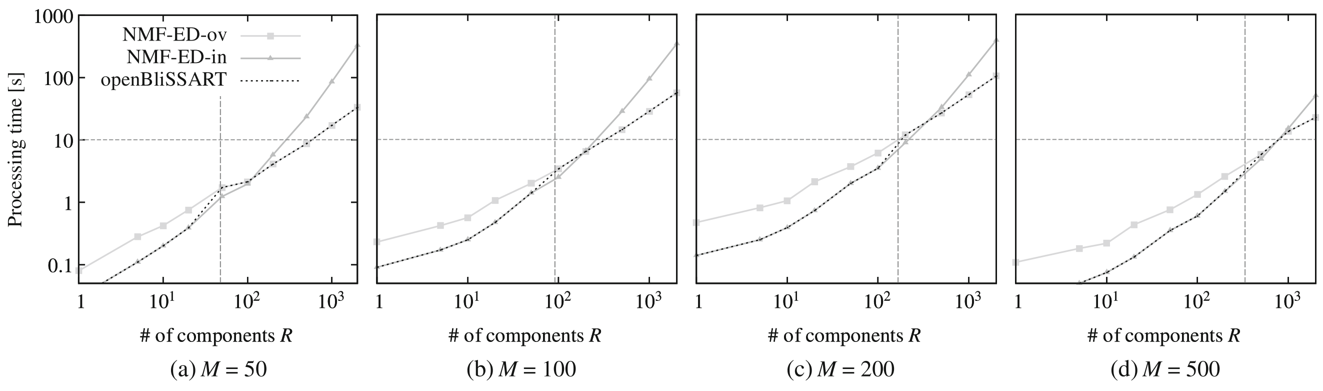
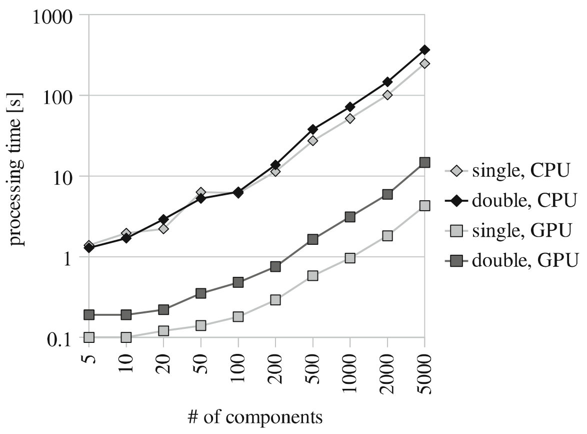


Figure 3 Euclidean NMF processing times depending on the number of components (1–2,000), for matrices with $N = 1{,}000$ columns and $M \in \{50, 100, 200, 500\}$ rows (**a–d**); CPU computation. 'NMF-ED-ov' and 'NMF-ED-in' denote the algorithms optimized for overcomplete and incomplete factorization, according to Section 3.3. 'openBliSSART' refers to the proposed automatic selection between 'NMF-ED-ov' and 'NMF-ED-in' based on the criterion $R(M+N) > MN$ to distinguish over- from incomplete factorization. The limit case $R(M+N) = MN$ is shown by the *vertical bars*, and real-time capability (processing time <10 s) is indicated by the *horizontal lines*.



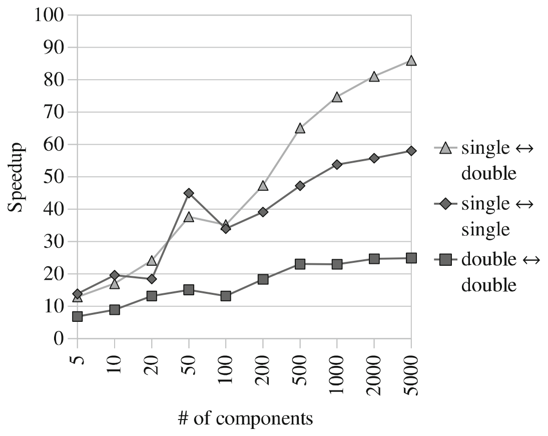


Figure 4 Processing time of the NMF algorithm (KL divergence) on a 500× 1,000 matrix by the number of NMF components; single/double floating point precision and CPU/GPU computation. Speedup for double or single precision on both CPU and GPU, and single precision GPU vs. double precision CPU.

Measuring the speedup by using parallel GPU instead of CPU computation, we observe a speedup of at least 6.8 even for ‘small’ matrices ($R = 5$), indicating that the overhead by parallel computation is quite low in a typical audio source separation scenario. Speedups of up to 24.9 are reached for double precision and up to 58.1 for single precision. In other words, the maximum speedup obtained by using single-precision GPU computation instead of double-precision CPU computation is 86.0. As expected, speedups increase with the dimensionality R. In the result, by using single-precision GPU computation, real-time processing (i. e., a processing time below 10 s) is achievable even for 5,000 NMF components. Note that the above benchmark results have been obtained with the openBliSSART benchmark tool delivered with the source code to ensure best reproducibility.

To put our results in perspective, we point out that in [8], a speedup factor of 28 has been reported for sparse NMF classification of a spectral matrix with $R = 8{,}000$, $M = 690$ and $N = 182$, comparing double precision NMF computations on a CPU with a single precision implementation on a GPU. In that study, a slightly less performant GPU with only 675 MHz core frequency (instead of 810) was used; however, it can be assumed that the timing of the CPU computation refers to using both cores of the employed Intel Core2Duo CPU with 2.4 GHz (as is the default setting in Matlab) while we used only one core. Generally, the RTFs reported in our study are higher than in [8], where supervised NMF is applied (i.e., only the **H** matrix is updated), while our performance measurements pertain to unsupervised NMF including the update for **W**. Furthermore, in an earlier study on parallelization of music source separation by NMF [1], an 18 × speedup has been reported for $R = 30$, $M = 512$ and $N = 3{,}445$ comparing single precision calculations in a single CPU thread (Intel Core i7 920) vs. a single precision CUDA implementation (NVIDIA GTX 280); this speedup is lower than the speedup gained in our experiments for similar dimensions, but this can be attributed to the slightly different processing hardware used. Overall, we conclude that our results corroborate the results from [8] and [1] in a larger scale study.

4.3 Benchmark Performances in Supervised Speech Separation

The remainder of our experimental evaluation is dedicated to a real-life application: supervised speaker separation (cf. Section 2.3). We defined our evaluation methodology in accordance with [24]: Since in [24], no significant gain in perceptual quality could be obtained by using NMD instead of NMF bases, we restrict the evaluation to NMF. For all subsequent experiments, we selected 12 random pairs of male and female speakers from the TIMIT database. For each pair, we mixed together two randomly selected sentences of roughly equal length, and computed an NMF basis **W** from the spectra in the other sentences spoken by each speaker using 250 multiplicative update iterations. Through supervised NMF with **W**, separated signals for both speakers were obtained. As quality measures, we employed signal-to-distortion ratio (SDR) as a measure of overall separation quality, source-to-interference ratio (SIR) to quantify suppression of the undesired speaker (which may however lead to information loss in the wanted speech signal due to spectral overlap), and source-to-artifact ratio (SAR) to evaluate degradation of speech quality by the separation.

Table 2 Double and single precision in supervised speech separation by NMF: separation quality in terms of SDR, SIR and SAR, as well as corresponding real-time factors (RTF) for CPU and GPU computation.

	Separation quality [dB]			RTF	
Precision	SDR	SIR	SAR	CPU	GPU
Double	5.16	10.15	7.92	0.522	0.068
Single	5.16	10.15	7.92	0.937	0.033

Measurements were carried out using the open-source BSS_Eval toolkit [28].

First, we determine whether using single instead of double floating point precision has a negative impact on separation quality. From Table 2, it is evident that this is not the case: In fact, the SDR, SIR and SAR values are identical for double and single precision up to the third decimal. This is in accordance with the findings of [8] for non-negative sparse classification. Interestingly, the matrix dimension in our case (50 components) coincides with a configuration where double precision is faster than single precision in CPU computation, confirming the singularity evident from Fig. 4; this surprising result has been corroborated in multiple repetitions to cope with random fluctuations due to operating system CPU usage etc. Conversely, in GPU computation, the RTF can be halved by using single precision without decreasing the separation quality.

In a second experiment, we assess the effect of using different numbers of iterations, DFT window sizes and the NMF cost function. The importance of the former two parameters on separation quality has been pointed out in [24], and various previous studies clearly suggest that different cost functions maybe optimal for different source separation problems [5, 30, 32]. Still, to our knowledge, the trade-off between separation quality and the RTF has been rarely investigated in the light of these parameters, although the algorithms minimizing different cost functions considerably differ in the number of required matrix operations, and their complexity (cf. Section 3.3).

In this experiment, the number of separation iterations was chosen from {20, 40, 80, 160, 320} due to the quick saturation of the convergence of multiplicative update NMF algorithms in audio source separation [21]. The different DFT window sizes considered were powers of two, ranging from 2^6 to 2^{12}, or 8–256 ms assuming 16 kHz sample rate. We evaluated the RTF for both CPU and GPU computation, taking the elapsed computation time over the length of the mixed signals. From Fig. 5, it can be seen that the best average results are obtained by using the KL divergence as cost function. The Euclidean distance allows faster separation at the expense of quality. Reasonable results are only achieved for long window sizes (256 ms), which limits the practical applicability in contexts where real-time operation is required. Finally, the IS divergence enables robust separation, but is inferior to KL divergence both in terms of separation quality and RTF.

Generally, we observe that in case of inadequate modeling of the sources (indicated by overall low SDR), more iterations do not necessarily improve separation quality, despite the fact that they linearly increase computational complexity; in fact, more iterations sometimes degrade quality, e. g., for the Euclidean cost function and 16 or 64 ms window size. As expected, the window size itself only slightly influences the RTF: Recall that the asymptotic complexity of each iteration step is $O(MNR)$; M is linear in the window size while N corresponds to the number of frames, and thus changing the window size while keeping the overlap between successive frames constant leaves the product MN unaffected.

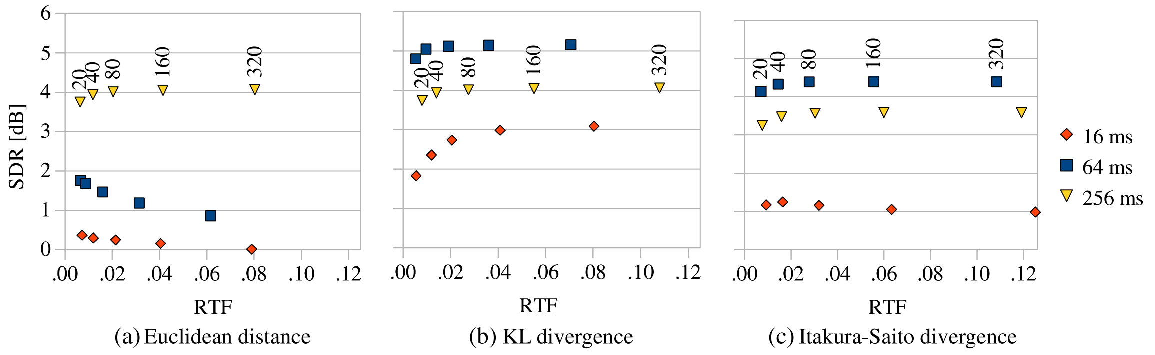


Figure 5 Tuning of the trade-off between RTF and perceptual quality for supervised NMF speech separation by adjusting the number of NMF iterations (20–320), the DFT window size (16, 64, 256 ms) and the NMF cost function. The average signal-to-distortion ratio (SDR) [28] is shown for the separation of mixed signals spoken by pairs of male/female speakers (24 speakers total) from the TIMIT database.

5 Conclusion

We have outlined recent developments in the openBliSSART toolkit for NMF-based audio source separation, speech enhancement and non-negative feature extraction. Besides numerical optimizations of the NMF algorithms themselves, we were able to drastically reduce the computational effort required in practice by using parallel computation on GPUs, reaching real-time capability for several thousand NMF components. Still, the exact speedup induced by parallel computation of NMF is depending on a variety of external factors such as the number of double and single precision floating point units which can operate in parallel, the implementation of the employed BLAS libraries and the instruction set architecture of the CPU and GPU. While separation quality is seemingly unaffected by floating point precision, using single precision is not necessarily faster for all configurations. Conducting performance studies in a real-life, end-user oriented setup featuring a state-of-the-art desktop PC—as in this article—usually implies that the CPU has a complex instruction set architecture (ISA) where performance of floating point operations strongly depends on the used machine instructions. This is in contrast to GPUs whose ISA is usually similar to the concept of reduced instruction set computer (RISC) architectures. Overall, we believe that by providing the source code of the algorithms as well as the benchmarks, it will be straightforward for the research community to gain additional insights into the performance of NMF in different hardware setups.

Besides, the observed real-time factors motivate us to address truly on-line, i. e., incremental audio processing in future development. It will be of particular research interest to study incremental refinement of source models and classification of on-line estimated components in semi-supervised separation.

Acknowledgement The authors would like to thank Alexander Lehmann for his contributions to the openBliSSART code.

References

1. Battenberg, E., & Wessel, D. (2009). Accelerating non-negative matrix factorization for audio source separation on multi-core and many-core architectures. In *Proc. of 10th International Society for Music Information Retrieval conference (ISMIR)* (pp. 501–506). Kobe, Japan.
2. Cichocki, A., Zdunek, R., Phan, A. H., & Amari, S. I. (2009). *Nonnegative matrix and tensor factorizations*. Wiley.
3. Cooke, M., Hershey, J. R., & Rennie, S. J. (2010). Monaural speech separation and recognition challenge. *Computer Speech and Language, 24*, 1–15.
4. Durrieu, J. L., Richard, G., David, B., & Févotte, C. (2010). Source/filter model for unsupervised main melody extraction from polyphonic audio signals. *IEEE Transactions on Audio, Speech, and Language Processing, 18*(3), 564–575.
5. Févotte, C., Bertin, N., & Durrieu, J. L. (2009). Nonnegative matrix factorization with the Itakura–Saito divergence: With application to music analysis. *Neural Computation, 21*(3), 793–830.
6. Frigo, M., & Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE, 93*(2), 216–231.
7. Gemmeke, J., Virtanen, T., & Hurmalainen, A. (2011). Exemplar-based sparse representations for noise robust automatic speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing, 19*(7), 2067–2080.
8. Gemmeke, J. F., Hurmalainen, A., Virtanen, T., & Sun, Y. (2011). Toward a practical implementation of exemplar-based noise robust ASR. In *Proc. of EUSIPCO* (pp. 1490–1494).
9. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter, 11*, 10–18.
10. Heittola, T., Mesaros, A., Virtanen, T., & Eronen, A. (2011). Sound event detection in multisource environments using source separation. In *Proc. of CHiME workshop* (pp. 86–90). Florence, Italy.
11. Helen, M., & Virtanen, T. (2005). Separation of drums from polyphonic music using non-negative matrix factorization and support vector machine. In *Proc. of EUSIPCO*. Antalya, Turkey.
12. Hurmalainen, A., Gemmeke, J., & Virtanen, T. (2011). Non-negative matrix deconvolution in noise robust speech recognition. In *Proc. of ICASSP* (pp. 4588–4591). Prague, Czech Republic.
13. Hyvärinen, A., Karhunen, J., & Oja, E. (2001). *Independent component analysis*. New York: Wiley.
14. Lee, D. D., & Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In *Proc. of NIPS* (pp. 556–562). Vancouver, Canada.
15. Mesaros, A., & Virtanen, T. (2009). Automatic recognition of lyrics in singing. *EURASIP Journal on Audio, Speech, and Music Processing*, Article ID 546047.
16. O'Grady, P. D., & Pearlmutter, B. A. (2007). Discovering convolutive speech phones using sparseness and non-negativity constraints. In *Proc. of ICA*. London, UK.
17. Ozerov, A., Févotte, C., & Charbit, M. (2009). Factorial scaled hidden Markov model for polyphonic audio representation and source separation. In *Proc. of WASPAA* (pp. 121–124). Mohonk, NY, United States.
18. Ozerov, A., & Vincent, E. (2011). Using the FASST source separation toolbox for noise robust speech recognition. In *Proc. of CHiME workshop* (pp. 86–87). Florence, Italy.
19. Raj, B., Virtanen, T., Chaudhuri, S., & Singh, R. (2010). Non-negative matrix factorization based compensation of music for automatic speech recognition. In *Proc. of Interspeech*. Makuhari, Japan.
20. Schmidt, M. N., & Olsson, R. K. (2006). Single-channel speech separation using sparse non-negative matrix factorization. In *Proc. of Interspeech*. Pittsburgh, PA, USA.
21. Schuller, B., Lehmann, A., Weninger, F., Eyben, F., & Rigoll, G. (2009). Blind enhancement of the rhythmic and harmonic sections by NMF: Does it help? In *Proc. of the international conference on acoustics (NAG/DAGA 2009)* (pp. 361–364). DEGA, Rotterdam, Netherlands.
22. Schuller, B., & Weninger, F. (2010). Discrimination of speech and non-linguistic vocalizations by non-negative matrix

factorization. In *Proc. of ICASSP* (pp. 5054–5057). Dallas, TX, USA.

23. Schuller, B., Weninger, F., Wöllmer, M., Sun, Y., & Rigoll, G. (2010). Non-negative matrix factorization as noise-robust feature extractor for speech recognition. In *Proc. of ICASSP* (pp. 4562–4565). Dallas, TX, USA.
24. Smaragdis, P. (2007). Convolutive speech bases and their application to supervised speech separation. *IEEE Transactions on Audio, Speech and Language Processing, 15*(1), 1–14.
25. Smaragdis, P., & Brown, J. C. (2003). Non-negative matrix factorization for polyphonic music transcription. In *Proc. of IEEE workshop on applications of signal processing to audio and acoustics* (pp. 177–180). New Paltz, NY, USA.
26. Smaragdis, P., Raj, B., & Shashanka, M. (2007). Supervised and semi-supervised separation of sounds from single-channel mixtures. In *Proc. of ICA* (pp. 414–421). Berlin: Springer.
27. Uhle, C., Dittmar, C., & Sporer, T. (2003). Extraction of drum tracks from polyphonic music using independent subspace analysis. In *Proc. of ICA*. Nara, Japan.
28. Vincent, E., Gribonval, R., & Févotte, C. (2006). Performance measurement in blind audio source separation. *IEEE Transactions on Audio, Speech and Language Processing, 14*(4), 1462–1469.
29. Virtanen, T. (2007). Monaural sound source separation by nonnegative matrix factorization with temporal continuity and sparseness criteria. *IEEE Transactions on Audio, Speech and Language Processing, 15*(3), 1066–1074.
30. Wang, W., Cichocki, A., & Chambers, J. A. (2009). A multiplicative algorithm for convolutive non-negative matrix factorization based on squared Euclidean distance. *IEEE Transactions on Signal Processing, 57*(7), 2858–2864.
31. Weninger, F., Geiger, J., Wöllmer, M., Schuller, B., & Rigoll, G. (2011). The Munich 2011 CHiME challenge contribution: NMF-BLSTM speech enhancement and recognition for reverberated multisource environments. In *Proc. of CHiME workshop* (pp. 24–29). Florence, Italy.
32. Weninger, F., Lehmann, A., & Schuller, B. (2011). openBliSSART: Design and evaluation of a research toolkit for blind source separation in audio recognition tasks. In *Proc. of ICASSP* (pp. 1625–1628). Prague, Czech Republic.
33. Weninger, F., Schuller, B., Batliner, A., Steidl, S., & Seppi, D. (2011). Recognition of nonprototypical emotions in reverberated and noisy speech by nonnegative matrix factorization. *EURASIP Journal on Advances in Signal Processing, Special Issue on Emotion and Mental State Recognition from Speech*, Article ID 838790, 16 pp.
34. Weninger, F., Schuller, B., Wöllmer, M., & Rigoll, G. (2011). Localization of non-linguistic events in spontaneous speech by non-negative matrix factorization and long short-term memory. In *Proc. of ICASSP* (pp. 5840–5843). Prague, Czech Republic.
35. Whaley, R. C., Petitet, A., & Dongarra, J. (2001). Automated empirical optimization of software and the ATLAS project. *Parallel Computing, 27*(1–2), 3–35.
36. Wilson, K. W., Raj, B., & Smaragdis, P. (2008). Regularized non-negative matrix factorization with temporal dependencies for speech denoising. In *Proc. of Interspeech*. Brisbane, Australia.
37. Yasuda, K. (2008). Accelerating density functional calculations with graphics processing unit. *Journal of Chemical Theory and Computation, 4*, 1230–1236.
38. Young, S. J., Evermann, G., Gales, M. J. F., Kershaw, D., Moore, G., Odell, J. J., et al. (2006). *The HTK book version 3.4*. Cambridge University Engineering Department, Cambridge, UK.

Felix Weninger received his diploma in computer science and electrical engineering from Technische Universität Müunchen (TUM) in 2009. He is currently pursuing his Ph. D. degree as a researcher in the Intelligent Audio Analysis Group, focusing on source separation, feature extraction and machine learning for real-life speech recognition and music information retrieval tasks.

Björn Schuller received his diploma in 1999 and his doctoral degree for his study on Automatic Speech and Emotion Recognition in 2006, both in electrical engineering and information technology from TUM. He is tenured as Senior Researcher and Lecturer in Pattern Recognition and Speech Processing heading the Intelligent Audio Analysis Group at TUM's Institute for Human-Machine Communication since 2006.