SDR — MEDIUM RARE WITH FAST COMPUTATIONS

Robin Scheibler

LINE Corporation, Tokyo, Japan

ABSTRACT

We revisit the widely used bss_eval metrics for source separation with an eye out for performance. We propose a fast algorithm fixing shortcomings of publicly available implementations. First, we show that the metrics are fully specified by the squared cosine of just two angles between estimate and reference subspaces. Second, large linear systems are involved. However, they are structured, and we apply a fast iterative method based on conjugate gradient descent. The complexity of this step is thus reduced by a factor quadratic in the distortion filter size used in bss_eval, usually 512. In experiments, we assess speed and numerical accuracy. Not only is the loss of accuracy due to the approximate solver acceptable for most applications, but the speed-up is up to two orders of magnitude in some, not so extreme, cases. We confirm that our implementation can train neural networks, and find that longer distortion filters may be beneficial.

Index Terms— source separation, performance evaluation, bss_eval, signal-to-distortion ratio, conjugate gradient descent

1. INTRODUCTION

Source separation (SS) refers to a family of technique that can be used to recover signals of interests from their mixtures using only minimal prior information. It has broad applications, but we focus on audio, e.g., for speech [1] and music [2] signals. SS comes in many flavors. Deep neural networks (DNN) have been successfully used to separate multiple speakers from a single microphone's signal [3]. Blind SS (BSS) approaches based on independent component analysis work on the determined case where there as many sources as measurements [4]. Convolutive mixtures, such as found in audio, may be handled by independent vector analysis (IVA) [5], [6]. Finally, overdetermined IVA tackles the case where redundant measurements are available [7], [8]. Performance evaluation is key to developing new algorithms and requires relevant metrics to be available. For audio SS, the *bss_eval* metrics, i.e., signal-to-distortion, interference, and artifact ratios (SDR, SIR, and SAR, respectively), are a de facto standard [9] and have been routinely used for the evaluation of new algorithms, e.g. [3], [7], [10], [11]. They decompose the estimated signals into orthogonal components corresponding to target sources and artifacts, as illustrated in Fig. 1. Some amount of distortion in the estimate is allowed by forgiving a 512 taps filter.

It has been argued that these filters may actually be detrimental, especially for mask-based approaches [12]. As a countermeasure, the scale-invariant SDR (SI-SDR), i.e., the SDR with a single tap filter, has been proposed [12]. Subsequently, it has been used for end-to-end training of separation networks [13], [14]. However, the classic bss_eval SDR has been recently vindicated and shown to outperform the SI-SDR as a loss for end-to-end training of linear separation systems [8]. Nevertheless, some computational challenges remain. Computation of the optimal filters involves inversion of very large matrices, with cubic complexity using direct solvers. This may be crippling if many short signals have to be evaluated, e.g. in utterance-level permutation invariant training (uPIT) [15]. Furthermore, publicly available implementations, such as in `mir_eval`, are not as performant as one would desire, leading to long computation times when applied to large datasets, especially when the number of channels increases. For iterative methods [7], [10], bss_eval metrics have to be evaluated at multiple iterations to assess convergence.

We propose a new, highly efficient algorithm for the computation of the bss_eval metrics, i.e. SDR, SIR, and SAR. First, we provide a finer analysis of the definition of the metrics, letting us reduce to a minimum the number of steps dealing with the full length of the signals. The resulting savings are substantial since audio signals are typically long. Second, to reduce computations for the distortion filters, we propose to use conjugate gradient descent (CGD) [16]. We implement the proposed algorithm (i.e., the `bss_eval_sources` routine) in `pytorch` [17], making it fully differentiable and GPU-enabled[1]. We analyze the trade-off between numerical accuracy and speed in experiments on speech signals. Our proposed implementation is orders of magnitude faster than publicly available ones. The simplified steps provide savings for longer signals, while the CGD kicks in for more channels, or longer filters. For SDR only computations, we show up to 27× speed-up for 8 channels and a 1024 taps filter compared to [8]. We demonstrate successful training of a neural network for source separation using our implementation of the SDR as the loss. Interestingly, we find that doubling the length of the filters (1024 taps) leads to further improvements.

2. BACKGROUND

Vectors and matrices are represented by bold lower and upper case letters, respectively. The norm of vector $\boldsymbol{x}$ is written $\|\boldsymbol{x}\| = (\boldsymbol{x}^\top \boldsymbol{x})^{1/2}$. The convolution of vectors $\boldsymbol{x}$ and $\boldsymbol{h}$ is denoted $\boldsymbol{x} \star \boldsymbol{h}$.

We consider the case where we have M estimated signals $\hat{\boldsymbol{s}}_m$. Each contains a convolutional mixture of K reference signals $\boldsymbol{s}_k$ and a component $\boldsymbol{b}_m$ of artifacts for which no reference is available,

$$\hat{\boldsymbol{s}}_m = \sum\nolimits_k \boldsymbol{h}_{km} \star \boldsymbol{s}_k + \boldsymbol{b}_m, \quad m = 1, \ldots, M, \tag{1}$$

where $\hat{\boldsymbol{s}}_m$, $\boldsymbol{s}_k$, and $\boldsymbol{b}_m$ are all real vectors of length T. The length of the impulse responses $\boldsymbol{h}_{km}$ is assumed to be short compared to T. For simplicity, the convolution operation here includes truncation to size T. In most cases, the number of estimates and references is the same, i.e. $M = K$. We keep them distinct for generality.

2.1. bss_eval v3.0

There exists a few variants of the bss_eval metrics [9], but we concentrate on the so-called v3.0. It is the most recent and the one

This paper is reproducible. Code and data are available at `http://github.com/fakufaku/sdr_medium_rare/`.

[1] At the tip of your fingers: `pip install fast-bss-eval`

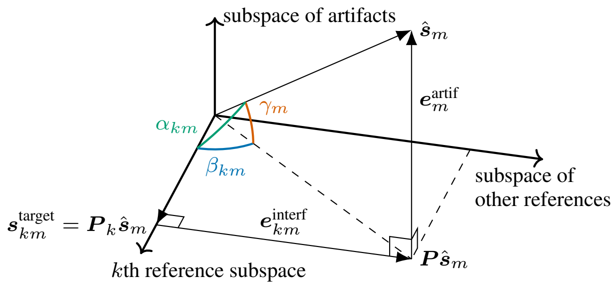

Fig. 1: Illustration of the decomposition operated by bss_eval. The estimated source $\hat{\boldsymbol{s}}_m$ is decomposed into $\boldsymbol{s}^{\text{target}}_{km}$, $\boldsymbol{e}^{\text{interf}}_{km}$, and $\boldsymbol{e}^{\text{artif}}_{km}$ by orthogonal projections onto the subspaces spanned by the L shifts of $\boldsymbol{s}_k$, of other references, and the artifacts, respectively. In Section 3.1, we show that SDR, SIR, and SAR are uniquely determined by angles α_{km}, β_{km}, and γ_{km}, respectively.

implemented in `mir_eval` [18] and `ci_sdr` [8]. In general, the matching of source estimates to their reference is not known and must be computed. To this end, the metrics must be computed for each pair $(\hat{\boldsymbol{s}}_m, \boldsymbol{s}_k)$ before the best matching is found.

bss_eval decomposes the estimated signal as shown in Fig. 1,

$$\boldsymbol{s}^{\text{target}}_{km} = \boldsymbol{P}_k\hat{\boldsymbol{s}}_m, \quad \boldsymbol{e}^{\text{interf}}_{km} = \boldsymbol{P}\hat{\boldsymbol{s}}_m - \boldsymbol{P}_k\hat{\boldsymbol{s}}_m, \quad \boldsymbol{e}^{\text{artif}}_{km} = \hat{\boldsymbol{s}}_m - \boldsymbol{P}\hat{\boldsymbol{s}}_m.$$

Matrices $\boldsymbol{P}_k$ and $\boldsymbol{P}$ are projection matrices onto the L shifts of $\boldsymbol{s}_k$ and of all references, respectively. Let $\boldsymbol{A}_k \in \mathbb{R}^{(T+L-1)\times L}$ contain the L shifts of $\boldsymbol{s}_k$ in its columns and $\boldsymbol{A} = [\boldsymbol{A}_1, \dots, \boldsymbol{A}_K]$, then

$$\boldsymbol{P}_k = \boldsymbol{A}_k(\boldsymbol{A}_k^\top \boldsymbol{A}_k)^{-1}\boldsymbol{A}_k^\top, \quad \boldsymbol{P} = \boldsymbol{A}(\boldsymbol{A}^\top \boldsymbol{A})^{-1}\boldsymbol{A}^\top. \tag{2}$$

Then, SDR, SIR, and SAR, in decibels, are defined as follows,

$$\text{SDR}_{km} = 10\log_{10}\frac{\|\boldsymbol{s}^{\text{target}}_{km}\|^2}{\|\boldsymbol{e}^{\text{interf}}_{km} + \boldsymbol{e}^{\text{artif}}_{km}\|^2}, \tag{3}$$

$$\text{SIR}_{km} = 10\log_{10}\frac{\|\boldsymbol{s}^{\text{target}}_{km}\|^2}{\|\boldsymbol{e}^{\text{interf}}_{km}\|^2}, \tag{4}$$

$$\text{SAR}_{km} = 10\log_{10}\frac{\|\boldsymbol{s}^{\text{target}}_{km} + \boldsymbol{e}^{\text{interf}}_{km}\|^2}{\|\boldsymbol{e}^{\text{artif}}_{m}\|^2}. \tag{5}$$

Finally, assuming for simplicity that $K = M$, the permutation of the estimated sources $\pi : \{1, \dots, K\} \to \{1, \dots, K\}$ that maximizes $\sum_k \text{SIR}_{k\,\pi(k)}$ is chosen[2].

2.2. Standard Implementations

Publicly available implementations of bss_eval [8], [9], [18] all follow a fairly straightforward approach for the computations. They all use a fixed or default value of $L = 512$. We start by defining the autocorrelation matrix of reference $\boldsymbol{s}_k$, of all references, and the cross-correlation of $\boldsymbol{s}_k$ and estimate $\hat{\boldsymbol{s}}_m$,

$$\mathbf{R}_k = \boldsymbol{A}_k^\top \boldsymbol{A}_k, \quad \mathbf{R} = \boldsymbol{A}^\top \boldsymbol{A}, \quad \boldsymbol{x}_{km} = \boldsymbol{A}_k^\top \hat{\boldsymbol{s}}_m,$$

respectively. Then, SDR, SIR, and SAR are computed as follows.

1. Compute $\mathbf{R}$ and $\boldsymbol{x}_{km}$ for all k and m. The former is a block-Toeplitz matrix containing $\mathbf{R}_k$ as its diagonal blocks. The complexity is $O(K^2 T \log T)$ and $O(KMT \log T)$, respectively, using the fast Fourier transform (FFT) [19].
2. Compute filters $\boldsymbol{h}_{km}$ in $O(KL^3 + KML^2)$ by solving
$$\mathbf{R}_k[\boldsymbol{h}_{k1}, \dots, \boldsymbol{h}_{kM}] = [\boldsymbol{x}_{k1}, \dots, \boldsymbol{x}_{kM}]. \tag{6}$$
3. Compute filters $\boldsymbol{g}_{k\ell}$ in $O(K^3L^3 + MK^2L^2)$ by solving
$$\mathbf{R}\begin{bmatrix} \boldsymbol{g}_{11} & \cdots & \boldsymbol{g}_{1K} \\ \vdots & \ddots & \vdots \\ \boldsymbol{g}_{K1} & \cdots & \boldsymbol{g}_{KK}\end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_{11} & \cdots & \boldsymbol{x}_{1M} \\ \vdots & \ddots & \vdots \\ \boldsymbol{x}_{K1} & \cdots & \boldsymbol{x}_{KM}\end{bmatrix}. \tag{7}$$
4. Compute $\boldsymbol{s}^{\text{target}}_{km} = \boldsymbol{h}_{km} \star \boldsymbol{s}_k$ in $O(KMT \log L)$.
5. Compute $\boldsymbol{u}_m = \sum_k \boldsymbol{g}_{km} \star \boldsymbol{s}_k$ in $O(KMT \log L)$.
6. Compute $\boldsymbol{e}^{\text{interf}}_{km} = \boldsymbol{s}^{\text{target}}_{km} - \boldsymbol{u}_m$ and $\boldsymbol{e}^{\text{artif}}_{m} = \hat{\boldsymbol{s}}_m - \boldsymbol{u}_m$. Then, compute the SDR_{km}, SIR_{km}, and SAR_{km} according to (3), (4), and (5). The complexity is $O(KMT)$.
7. Find the best permutation of the estimated sources in $O(M^3)$ by using the Hungarian algorithm[3] [23].

3. PROPOSED ALGORITHM

We identified the following inefficiencies in the implementation described above. Components $\boldsymbol{s}^{\text{target}}$, $\boldsymbol{e}^{\text{interf}}$, and $\boldsymbol{e}^{\text{artif}}$ do not need to be computed. We show below that we can replace steps 4, 5, and 6 by a simpler computation. These may not seem very computationally expensive, however, they operate on the full length T of the input signals. Audio signals may be long, e.g., 30 s sampled at 16 kHz is $T = 480\,000$ samples. The linear systems (6) and (7) are expensive to solve directly due to L being typically large, e.g., $L = 512$. However, they are Toeplitz and block-Toeplitz, respectively, and efficient algorithms can be applied.

3.1. Efficient Computation using Cosine Metrics

Since the bss_eval metrics are not sensitive to the scales of $\boldsymbol{s}_k$ and $\hat{\boldsymbol{s}}_m$, we will assume that all signals are scaled to have unit norm, i.e. $\|\boldsymbol{s}_k\| = \|\hat{\boldsymbol{s}}_m\| = 1$ for all k, m.

Definition 1 (Cosine Metrics). *We define the following new metrics,*

$$c_{km} = \hat{\boldsymbol{s}}_m^\top \boldsymbol{P}_k \hat{\boldsymbol{s}}_m = \boldsymbol{x}_{km}^\top \mathbf{R}_k^{-1}\boldsymbol{x}_{km} = \boldsymbol{x}_{km}^\top \boldsymbol{h}_{km}, \tag{8}$$

$$d_m = \hat{\boldsymbol{s}}_m^\top \boldsymbol{P}\hat{\boldsymbol{s}}_m = \boldsymbol{z}_m^\top \mathbf{R}^{-1}\boldsymbol{z}_m = \sum\nolimits_k \boldsymbol{x}_{km}^\top \boldsymbol{g}_{km}, \tag{9}$$

where $\boldsymbol{z}_m = \boldsymbol{A}^\top \hat{\boldsymbol{s}}_m = [\boldsymbol{x}_{1m}^\top, \dots, \boldsymbol{x}_{Km}^\top]^\top$.

Theorem 1. *The bss_eval metrics can be computed as follows,*

$$\text{SDR}_{km} = f(c_{km}), \tag{10}$$

$$\text{SIR}_{km} = f(c_{km}/d_m), \tag{11}$$

$$\text{SAR}_{m} = f(d_m). \tag{12}$$

where $f(x) = 10\log_{10}\left(\frac{x}{1-x}\right)$.

Proof. The proof follows directly from properties of projection matrices, namely idempotency and self-adjointness. Given a real projection operator $\mathbf{\Pi}$, these properties mean $\mathbf{\Pi}\mathbf{\Pi} = \mathbf{\Pi}$ and $\mathbf{\Pi} = \mathbf{\Pi}^\top$, respectively. Further, $\boldsymbol{I} - \mathbf{\Pi}$ is also a projection matrix. Thus,

$$\frac{\|\boldsymbol{s}^{\text{target}}_{km}\|^2}{\|\boldsymbol{e}^{\text{interf}}_{km} + \boldsymbol{e}^{\text{artif}}_{km}\|^2} = \frac{\|\boldsymbol{P}_k\hat{\boldsymbol{s}}_m\|^2}{\|(\boldsymbol{I} - \boldsymbol{P}_k)\hat{\boldsymbol{s}}_m\|^2} = \frac{\hat{\boldsymbol{s}}_m^\top \boldsymbol{P}_k\hat{\boldsymbol{s}}_m}{\hat{\boldsymbol{s}}_m^\top\hat{\boldsymbol{s}}_m - \hat{\boldsymbol{s}}_m^\top \boldsymbol{P}_k\hat{\boldsymbol{s}}_m},$$

[2] `ci_sdr` [8] uses the SDR since it does not compute the SIR.

[3] While the use of the Hungarian algorithm seems to have been rediscovered recently, it has long been applied in the BSS literature, e.g., [20]–[22].

and (10) follows since $c_{km} = \hat{\boldsymbol{s}}_m^\top \boldsymbol{P}_k \hat{\boldsymbol{s}}_m$ and we assumed $\|\hat{\boldsymbol{s}}_k\| = 1$. From their definition in (2), it is clear that the range space of $\boldsymbol{P}$ contains that of $\boldsymbol{P}_k$, and thus, $\boldsymbol{P}\boldsymbol{P}_k = \boldsymbol{P}_k\boldsymbol{P} = \boldsymbol{P}_k$. This can be used to show that $\boldsymbol{P} - \boldsymbol{P}_k$ is also a projection matrix. Thus,

$$\|\boldsymbol{e}_{km}^{\text{interf}}\|^2 = \|(\boldsymbol{P} - \boldsymbol{P}_k)\hat{\boldsymbol{s}}_m\|^2 = \hat{\boldsymbol{s}}_m^\top \boldsymbol{P} \hat{\boldsymbol{s}}_m - \hat{\boldsymbol{s}}_m^\top \boldsymbol{P}_k \hat{\boldsymbol{s}}_m, \quad (13)$$

and (11) follows by $d_m = \hat{\boldsymbol{s}}_m^\top \boldsymbol{P} \hat{\boldsymbol{s}}_m$. Finally, it can be seen that $\boldsymbol{s}_{km}^{\text{target}} + \boldsymbol{e}_{km}^{\text{interf}} = \boldsymbol{P}\hat{\boldsymbol{s}}_k$ and thus (12) follows similarly. $\square$

We can make a few observations. Once the filters $\boldsymbol{h}_{km}$ and $\boldsymbol{g}_{km}$ have been computed, c_{km} and d_m only require an extra $O(KML)$ operations. The SAR does not depend on the reference index k. We call the cosine metrics thus because they are the squared cosine of angles α_{km} and γ_m in Fig. 1. Moreover, c_{km}/d_m is the square cosine of β_{km}. A consequence is that the SI-SDR can be implemented simply as the square inner product of the normalized estimate and reference signals through the function $f(x)$.

3.2. Efficient Toeplitz Linear System Solver

We have established via Theorem 1 that efficiently solving the Toeplitz and block-Toeplitz systems (6) and (7) is key to the computation of the bss_eval metrics. Direct solution by Gaussian elimination has cubic time in the matrix size. However, highly efficient solvers are typically available in numerical linear algebra libraries such as BLAS and Lapack. The celebrated Levinson-Durbin recursion [24] works in quadratic time, but is seldom available in libraries. There exists also an even better alternative.

The CGD algorithm with a circulant preconditioner has complexity $O(L \log L)$ for an $L \times L$ Toeplitz system [16]. We briefly review the method here applied to solving (6). CGD only requires matrix-vector multiplication by the system matrix $\mathbf{R}_k$. For a Toeplitz matrix, such as $\mathbf{R}_k$, this can be done in $O(L \log L)$ operations by leveraging the FFT. Convergence of CGD is dictated by the distribution of eigenvalues of $\mathbf{R}_k$, and can be improved by a preconditioner. For example, the optimal circulant matrix $\boldsymbol{C}_k$ minimizing $\|\boldsymbol{C}_k - \mathbf{R}_k\|_F^2$ [25]. For symmetric $\mathbf{R}_k$ with first column $\boldsymbol{r} = [r_1, \ldots, r_L]^\top$, the first column of $\boldsymbol{C}_k$ is given by $(\boldsymbol{C}_k)_{11} = r_1$,

$$(\boldsymbol{C}_k)_{\ell 1} = L^{-1}\left[(L - \ell + 1) r_\ell + (\ell - 1) r_{L-\ell+1}\right], \; \ell \geq 2. \quad (14)$$

It has been shown that the eigenvalues of $\boldsymbol{C}_k^{-1}\mathbf{R}_k$ cluster around 1, and only a few iterations are required until convergence, independent of the matrix size. Multiplication by $\boldsymbol{C}_k^{-1}$ is done in $O(L \log L)$ time via the FFT. For K systems, the cost is thus $O(KL \log L)$.

For the block-Toeplitz system (7), we construct the preconditioner by replacing the Toeplitz blocks of $\mathbf{R}$ by their optimal circulant approximation. The formula is slightly different than (14) because the off-diagonal blocks are not symmetric. It can be found in [25]. By applying the FFT, we obtain a block-diagonal matrix with L $K \times K$ blocks that we invert with a direct solver. This is a one time cost of $O(LK^3)$. Matrix-vector multiplication by $\mathbf{R}$ or the preconditioner requires $O(K^2 L \log L)$ operations using the FFT. Thus, solving (7) requires $O(K^3 L + K^2 L \log L)$, a substantial saving by a factor L^2 compared to the direct method. We also found in practice that the solution of (6) is a good initial value to solve (7).

4. EXPERIMENTS

We assess the proposed implementation with respect to other publicly available Python implementations. **mir_eval**[4] [18] is the most widely used implementation and is regression tested to ensure the same output as the original Matlab implementation[5]. **sigsep**[6] is a recent re-implementation of the `mir_eval` implementation with a focus on performance. **ci_sdr**[7] implements the SDR only, but in a differentiable way, to be used to train neural networks. Experiments where conducted on a Linux workstation with an Intel® Xeon® Gold 6230 CPU 2.10 GHz with 8 cores, an NVIDIA® Tesla® V100 graphical processing unit (GPU), and 64 GB of RAM. Throughout, we use `solve` and `CGD10` to denote Gaussian elimination and 10 iterations of CGD, respectively. We use fp32 and fp64 to mean single and double precision floating-point modes, respectively.

Dataset We use the dataset of reverberant noisy speech mixtures introduced in [14]. The relative SNR of sources is selected at random in the range -5 dB to 5 dB. Speech and noise samples were selected from the WSJ1 [26] and CHIME3 datasets [27], respectively. Noise is scaled to obtain a final SNR between 10 dB to 30 dB. Mixtures contain two, three, and four sources, with an equal number of microphones. For each number of sources, the dataset is split into training, validation, and test with 37 416, 503, and 333 mixtures, respectively.

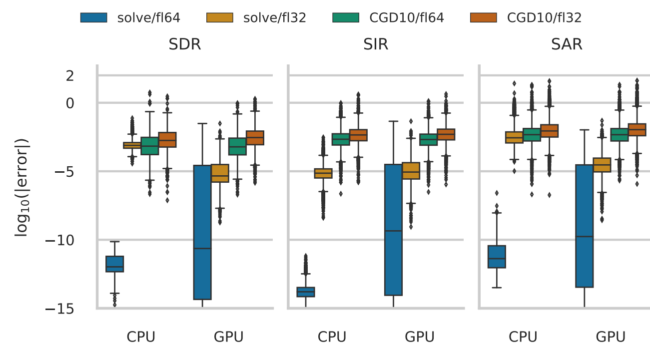

Fig. 2: Box-plots of the log-absolute error relative to `mir_eval` output.

4.1. Evaluation on Speech Mixtures Dataset

Our first experiment compares our proposed implementation to `mir_eval` and `sigsep` for the computation of SDR, SIR, and SAR. We use the test set, augmented by the output of separation by AuxIVA [10], doubling the number of samples. We measure the difference with respect to `mir_eval`'s output and the runtime.

Fig. 2 shows box-plots of the absolute difference with the output of `mir_eval`. Using `solve` with fp64 shows very little difference with `mir_eval`, less than 10^{-6} dB in all cases on the CPU. On the GPU, there is more variance, but, essentially, the error in decibels is negligible. Switching to fp32, the error is still small, between 10^{-5} dB to 10^{-3} dB, with the exception of SAR on the GPU, where a few outliers exceed 1 dB. When the linear systems are solved approximately with `CGD10`, the median error is below 10^{-2} dB. There are some outliers with error above 1 dB, more so for SAR than other metrics. We noted that the errors were zero-mean and thus do not impact averages over many samples. Using fp32 or fp64 did not make a big difference when using CGD.

Fig. 3 shows runtimes averaged over the whole dataset. We show results using CPU with 1 core (1CPU), 8 cores (8CPU), and GPU. In all cases the proposed implementation brings significant speed-up

[4] `https://github.com/craffel/mir_eval`

[5] `http://bass-db.gforge.inria.fr/bss_eval/`

[6] `https://github.com/sigsep/bsseval`

[7] `https://github.com/fgnt/ci_sdr`

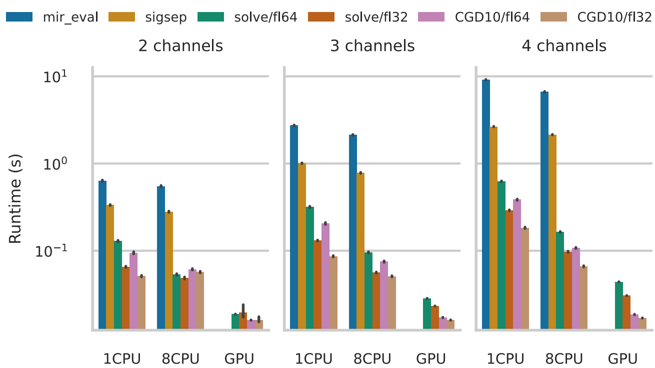

Fig. 3: Average runtime (s) to compute SDR, SIR, and SAR over the full test dataset. From left to right, 2, 3, and 4 channel signals. Within each subplot, we give runtimes for CPU with 1 core, 8 cores, and GPU.

compared to `mir_eval` and `sigsep`. There is about one order of magnitude speed up for two and three channels, and two orders for four channels. About one and two orders of magnitude difference for three and four channels, respectively. Going to the GPU brings yet another major speed-up. `mir_eval` and `sigsep` seem to benefit less from multiple cores, which we attribute to `numpy` being less efficient in this area than `pytorch`. For two channels using 8 CPU cores, `solve` was faster than CGD. In other cases, CGD is two to three times faster. This advantage is more salient on GPU.

4.2. Effects of Signal and Filter Length on Runtime

We analyze here the runtime behavior for SDR only computation for varying signal and filter lengths, and number of channels. We compare to the `ci_sdr` toolbox [8] which provides an SDR only implementation also based on `pytorch`. The measurement are done by computing the SDR for random signals on the GPU. Table 1 shows the average runtime of 10 measurements, each for a batch computation with 10 signals. When using `solve`, while the difference with `ci_sdr` is small for two channels, the gap steadily widens with the number of channels. The speed-up is also larger for longer signals, which is expected since the proposed implementation reduces computations involving the full length to a minimum. Next, we confirm that using CGD leads to a large speed gain. Most notably, doubling the filter size has no visible effect on the runtime for CGD, whereas it quadruples when using `solve`. The most extreme speed-ups occur for short signals and long filters, e.g., 10× to 27× depending on the number of channels. This may be very valuable for methods where PIT is applied to many short signals, such as in uPIT [15].

4.3. Training Neural Networks

Finally, we demonstrate the suitability of the proposed implementation for the training of separation networks. We train neural source models for determined source separation using AuxIVA as described in [14]. We compare the following loss functions, SI-SDR, as in [14], SDR with $L = 512$/`solve`, as in [8], $L = 512$/CGD10, and $L = 1024$/CGD10. The dataset is the one described above. We use the reverberant image sources as target signals. Algorithm and training details are described in [14], but the DNN used is the one from [8]. We train for 57 epochs with initial learning rate of 3×10^{-4}. Training is done on two sources mixtures, test on two, three, and four sources mixtures.

Table 1: Runtime in ms (speed-up) of the proposed method compared to the `ci_sdr` [8] implementation. Runtimes are for batches of 10 signals on GPU.

Filter length		512 taps		1024 taps	
Signal length		5 s	20 s	5 s	20 s
2 ch.	`ci_sdr`	27	43	87	104
	`solve`	21 (1×)	26 (2×)	67 (1×)	71 (1×)
	`CGD10`	9 (3×)	15 (3×)	9 (10×)	15 (7×)
3 ch.	`ci_sdr`	44	80	144	181
	`solve`	25 (2×)	38 (2×)	84 (2×)	97 (2×)
	`CGD10`	10 (4×)	22 (4×)	10 (14×)	23 (8×)
4 ch.	`ci_sdr`	72	136	233	297
	`solve`	29 (2×)	50 (3×)	92 (3×)	112 (3×)
	`CGD10`	12 (6×)	33 (4×)	12 (19×)	33 (9×)
5 ch.	`ci_sdr`	97	196	330	427
	`solve`	34 (3×)	65 (3×)	101 (3×)	132 (3×)
	`CGD10`	15 (6×)	46 (4×)	14 (22×)	46 (9×)
6 ch.	`ci_sdr`	129	268	447	590
	`solve`	41 (3×)	86 (3×)	119 (4×)	163 (4×)
	`CGD10`	18 (7×)	62 (4×)	18 (25×)	62 (9×)
7 ch.	`ci_sdr`	168	386	584	778
	`solve`	48 (3×)	108 (4×)	134 (4×)	193 (4×)
	`CGD10`	22 (8×)	82 (5×)	22 (26×)	82 (9×)
8 ch.	`ci_sdr`	208	462	741	987
	`solve`	54 (4×)	134 (3×)	148 (5×)	226 (4×)
	`CGD10`	27 (8×)	106 (4×)	27 (27×)	106 (9×)

Table 2: Mean SDR (dB) / WER (%) for determined source separation with a neural source model trained with the SDR as loss using different parameters. For evaluation, the SDR uses filter length $L = 512$ taps and `solve`.

Solver	L	2 ch.	3 ch.	4 ch.
(SI-SDR)	1	11.26 / 31.58	8.48 / 44.16	6.44 / 54.92
`solve`	512	11.50 / 30.19	8.76 / 42.62	6.61 / 54.83
`CGD10`	512	11.50 / 30.18	8.76 / 42.65	6.61 / 54.88
`CGD10`	1024	11.60 / 29.42	8.95 / 41.95	6.92 / 51.47

Table 2 shows the test results for the models with smallest validation loss. We evaluate in terms of SDR ($L = 512$/`solve`) and word error rate (WER) of an ASR model pre-trained using the `wsj/asr1` recipe of ESPNet [28]. First, we note that using the approximate CGD solver has no effect on the final accuracy. The results are in fact remarkably similar. Then, as in [8], we observe that allowing longer distortion filters with $L > 1$ leads to improvements of both SDR and WER. In fact, we observe that $L = 512$ might still be too short as $L = 1024$ leads to better performance.

5. CONCLUSION

We introduced an improved algorithm to implement the widely used bss_eval metrics for blind source separation evaluation. First, we reduce to a minimum computations that depend on the full length of input signals. Second, we propose to use an iterative solver to find the optimal distortion filters. We find very large runtime reductions that can potentially reduce the evaluation time from days to hours in SS experiments. The loss of accuracy due to the iterative solver does not impact average evaluation on datasets. Furthermore, it opens the door to using longer distortion filters. Experimental results suggest this may be beneficial to train separation networks. It also makes it possible to use bss_eval on signals sampled at a higher frequency, e.g., 44 kHz. We release our implementation as a Python package that can be used with both `numpy` and `pytorch`.

6. REFERENCES

[1] S. Makino, Ed., *Audio Source Separation*, ser. Signals and Communication Technology. Cham: Springer International Publishing, Jan. 2018.

[2] E. Cano, D. FitzGerald, A. Liutkus, M. D. Plumbley, and F.-R. Stöter, "Musical Source Separation: An Introduction," *IEEE Signal Process. Mag.*, vol. 36, no. 1, pp. 31–40, Jan. 2019.

[3] J. R. Hershey, Z. Chen, J. Le Roux, and S. Watanabe, "Deep clustering: Discriminative embeddings for segmentation and separation," in *Proc. IEEE ICASSP*, Shanghai, CN, Mar. 2016, pp. 31–35.

[4] P. Comon and C. Jutten, *Handbook of blind source separation: independent component analysis and applications ; 1st ed.* Oxford, UK: Academic Press/Elsevier, Jan. 2010.

[5] T. Kim, T. Eltoft, and T.-W. Lee, "Independent vector analysis: An extension of ICA to multivariate components," in *Advances in Cryptology — ASIACRYPT 2016*, vol. 3889, Berlin, Heidelberg: Springer Berlin Heidelberg, Jan. 2006, pp. 165–172.

[6] A. Hiroe, "Solution of permutation problem in frequency domain ICA, using multivariate probability density functions," in *Advances in Cryptology — ASIACRYPT 2016*, vol. 3889, Berlin, Heidelberg: Springer Berlin Heidelberg, Jan. 2006, pp. 601–608.

[7] R. Scheibler and N. Ono, "Independent vector analysis with more microphones than sources," in *Proc. IEEE WASPAA*, New Paltz, NY, USA, Oct. 2019, pp. 185–189.

[8] C. Boeddeker et al., "Convolutive transfer function invariant SDR training criteria for multi-channel reverberant speech separation," in *Proc. IEEE ICASSP*, Toronto, CA, Jun. 2021, pp. 8428–8432.

[9] E. Vincent, R. Gribonval, and C. Fevotte, "Performance measurement in blind audio source separation," *IEEE Trans. Audio Speech Lang. Process.*, vol. 14, no. 4, pp. 1462–1469, Jun. 2006.

[10] N. Ono, "Stable and fast update rules for independent vector analysis based on auxiliary function technique," in *Proc. IEEE WASPAA*, New Paltz, NY, USA, Oct. 2011, pp. 189–192.

[11] F.-R. Stöter, S. Uhlich, A. Liutkus, and Y. Mitsufuji, "Open-Unmix — a reference implementation for music source separation," *J. Open Source Softw.*, vol. 4, no. 41, p. 1667, Sep. 2019.

[12] J. Le Roux, S. Wisdom, H. Erdogan, and J. R. Hershey, "SDR – half-baked or well done?" In *Proc. IEEE ICASSP*, Brighton, UK, May 2019.

[13] Y. Luo and N. Mesgarani, "TaSNet: Time-domain audio separation network for real-time, single-channel speech separation," in *Proc. IEEE ICASSP*, Calgary, CA, Apr. 2018, pp. 696–700.

[14] R. Scheibler and M. Togami, "Surrogate source model learning for determined source separation," in *Proc. IEEE ICASSP*, Barcelona, ES, Jun. 2021, pp. 176–180.

[15] M. Kolbaek, D. Yu, Z.-H. Tan, and J. Jensen, "Multitalker speech separation with utterance-level permutation invariant training of deep recurrent neural networks," *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 25, no. 10, pp. 1901–1913, Aug. 2017.

[16] R. H. Chan and M. K. Ng, "Conjugate gradient methods for Toeplitz systems," *SIAM Review*, vol. 38, no. 3, pp. 427–482, Sep. 1996.

[17] A. Paszke et al., "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019.

[18] C. Raffel et al., "mir_eval: A transparent implementation of common MIR metrics," in *Proc. ISMIR*, Taipei, TW, Oct. 2014.

[19] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of computation*, pp. 297–301, Jan. 1965.

[20] L. Drake, A. Katsaggelos, J. Rutledge, and J. Zhang, "Sound source separation via computational auditory scene analysis-enhanced beamforming," in *Proc. IEEE SAM*, Aug. 2002, pp. 259–263.

[21] A. Ciaramella and R. Tagliaferri, "Amplitude and permutation indeterminacies in frequency domain convolved ICA," in *Proc. Int. Jt. Conf. Neural Netw.*, vol. 1, Jul. 2003, pp. 708–713.

[22] P. Tichavský and Z. Koldovský, "Optimal pairing of signal components separated by blind techniques," *IEEE Signal Process. Lett.*, vol. 11, no. 2, pp. 119–122, Feb. 2004.

[23] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, Mar. 1955.

[24] N. Levinson, "The Wiener RMS error criterion in filter design and prediction," *J. Math. Phys.*, vol. 25, no. 1-4, pp. 261–278, Apr. 1946.

[25] T. F. Chan, "An optimal circulant preconditioner for Toeplitz systems," *SIAM J. Sci. Stat. Comput.*, vol. 9, no. 4, pp. 766–771, Jan. 1988.

[26] Linguistic Data Consortium, and NIST Multimodal Information Group, *CSR-II (WSJ1) complete LDC94S13A*, Web Download, Linguistic Data Consortium, Philadelphia, 1994.

[27] J. Barker, R. Marxer, E. Vincent, and S. Watanabe, "The third 'CHiME' speech separation and recognition challenge: Dataset, task and baselines," in *Proc. IEEE ASRU*, Scottsdale, AZ, USA, Nov. 2015, pp. 504–511.

[28] S. Watanabe et al., "ESPnet: End-to-end speech processing toolkit," in *Proc. ISCA INTERSPEECH*, Hyderabad, IN, 2018, pp. 2207–2211.