Asteroid: the PyTorch-based audio source separation toolkit for researchers

Manuel Pariente, Samuele Cornell, Joris Cosentino, Sunit Sivasankaran, Efthymios Tzinis, Jens Heitkaemper, Michel Olvera, Fabian-Robert Stöter, Mathieu Hu, Juan M. Martín-Doñas, et al.

To cite this version:

Manuel Pariente, Samuele Cornell, Joris Cosentino, Sunit Sivasankaran, Efthymios Tzinis, et al.. Asteroid: the PyTorch-based audio source separation toolkit for researchers. Interspeech 2020, Oct 2020, Shanghai, China. hal-02962964

HAL Id: hal-02962964
https://inria.hal.science/hal-02962964

Submitted on 9 Oct 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Asteroid: the PyTorch-based audio source separation toolkit for researchers

Manuel Pariente\textsuperscript{1}, Samuele Cornell\textsuperscript{2}, Joris Cosentino\textsuperscript{1}, Sunit Sivasankaran\textsuperscript{1}, Efthymios Tzinis\textsuperscript{3}, Jens Heitkaemper\textsuperscript{2}, Michel Olvera\textsuperscript{1}, Fabian-Robert Stöter\textsuperscript{7}, Mathieu Hu\textsuperscript{1}, Juan M. Martín-Doñas\textsuperscript{6}, David Ditter\textsuperscript{3}, Ariel Frank\textsuperscript{8}, Antoine Deleforge\textsuperscript{1}, Emmanuel Vincent\textsuperscript{1}

\textsuperscript{1}Université de Lorraine, CNRS, Inria, LORIA, France  \textsuperscript{2}Università Politecnica delle Marche, Italy  \textsuperscript{3}University of Illinois at Urbana-Champaign, USA  \textsuperscript{4}Universität Paderborn, Germany  \textsuperscript{5}Inria and LIRMM, University of Montpellier, France  \textsuperscript{6}Universidad de Granada, Spain  \textsuperscript{7}Universität Hamburg, Germany  \textsuperscript{8}Technion - Israel Institute of Technology, Israel

https://github.com/mpariente/asteroid

Abstract

This paper describes Asteroid, the PyTorch-based audio source separation toolkit for researchers. Inspired by the most successful neural source separation systems, it provides all neural building blocks required to build such a system. To improve reproducibility, Kaldi-style recipes on common audio source separation datasets are also provided. This paper describes the software architecture of Asteroid and its most important features. By showing experimental results obtained with Asteroid’s recipes, we show that our implementations are at least on par with most results reported in reference papers. The toolkit is publicly available at github.com/mpariente/asteroid

Index Terms: source separation, speech enhancement, open-source software, end-to-end

1. Introduction

Audio source separation, which aims to separate a mixture signal into individual source signals, is essential to robust speech processing in real-world acoustic environments [1]. Classical open-source toolkits such as FASSST [2], HARK [3], ManyEars [4] and openBlISSART [5] which are based on probabilistic modelling, non-negative matrix factorization, sound source localization and/or beamforming have been successful in the past decade. However, they are now largely outperformed by deep learning-based approaches, at least on the task of single-channel source separation [6–10].

Several open-source toolkits have emerged for deep learning-based source separation. These include nussl (Northwestern University Source Separation Library) [11], ONSSEN (An Open-source Speech Separation and Enhancement Library) [12], Open-Unmix [13], and countless isolated implementations replicating some important papers.[1]

Both nussl and ONSSEN are written in PyTorch [14] and provide training and evaluation scripts for several state-of-the-art methods. However, data preparation steps are not provided and experiments are not easily configurable from the command line. Open-Unmix does provide a complete pipeline from data preparation until evaluation, but only for the Open-Unmix model on the music source separation task. Regarding the isolated implementations, some of them only contain the model, while others provide training scripts but assume that training data has been generated. Finally, very few provide the complete pipeline. Among the ones providing evaluation scripts, differences can often be found, e.g., discarding short utterances or splitting utterances in chunks and discarding the last one.

This paper describes Asteroid (Audio source separation on Steroids), a new open-source toolkit for deep learning-based audio source separation and speech enhancement, designed for researchers and practitioners. Based on PyTorch, one of the most widely used dynamic neural network toolkits, Asteroid is meant to be user-friendly, easily extensible, to promote reproducible research, and to enable easy experimentation. As such, it supports a wide range of datasets and architectures, and comes with recipes reproducing some important papers. Asteroid is built on the following principles:

1. Abstract only where necessary, i.e., use as much native PyTorch code as possible.
2. Allow importing third-party code with minimal changes.
3. Provide all steps from data preparation to evaluation.
4. Enable recipes to be configurable from the command line.

We present the audio source separation framework in Section\textsuperscript{2}. We describe Asteroid’s main features in Section\textsuperscript{3} and their implementation in Section\textsuperscript{4}. We provide example experimental results in Section\textsuperscript{5} and conclude in Section\textsuperscript{6}.

2. General framework

While Asteroid is not limited to a single task, single-channel source separation is currently its main focus. Hence, we will only consider this task in the rest of the paper. Let $x$ be a single channel recording of $J$ sources in noise:

$$x(t) = \sum_{j=1}^{J} s_j(t) + n(t),$$

(1)
where \( \{s_j\}_{j=1..J} \) are the source signals and \( n \) is an additive noise signal. The goal of source separation is to obtain source estimates \( \{\hat{s}_j\}_{j=1..J} \) given \( x \).

Most state-of-the-art neural source separation systems follow the encoder-masker-decoder approach depicted in Fig. 3. The encoder computes a short-time Fourier transform (STFT)-like representation \( X \) by convolving the time-domain signal \( x \) with an analysis filterbank. The representation \( X \) is fed to the masker network that estimates a mask for each source. The masks are then multiplied entrywise with \( X \) to obtain source estimates \( \{\hat{s}_j\}_{j=1..J} \) in the STFT-like domain. The time-domain source estimates \( \{\hat{s}_j\}_{j=1..J} \) are finally obtained by applying transposed convolutions to \( \{\hat{s}_j\}_{j=1..J} \) with a synthesis filterbank. The three networks are jointly trained using a loss function computed on the masks or their embeddings [6] [17] [18], on the STFT-like domain estimates [7] [15] [19], or directly on the time-domain estimates [8] [10] [16] [20].

3. Functionality

Asteroid follows the encoder-masker-decoder approach, and provides various choices of filterbanks, masker networks, and loss functions. It also provides training and evaluation tools and recipes for several datasets. We detail each of these below.

3.1. Analysis and synthesis filterbanks

As shown in [20] [23], various filterbanks can be used to train end-to-end source separation systems. A natural abstraction is to separate the filterbank object from the encoder and decoder objects. This is what we do in Asteroid. All filterbanks inherit from the Filterbank class. Each Filterbank can be combined with an Encoder or a Decoder, which respectively follow the nn.Conv1d and nn.ConvTranspose1d interfaces from PyTorch for consistency and ease of use. Notably, the STFTFB filterbank computes the STFT using simple convolutions, and the default filterbank matrix is orthogonal.

Asteroid supports free filters [8] [19], discrete Fourier transform (DFT) filters [19] [21], analytic free filters [19], improved parameterized sinc filters [22] [24] and the multi-phase Gammatone filterbank [23]. Automatic pseudo-inverse computation and dynamic filters (computed at runtime) are also supported. Because some of the filterbanks are complex-valued, we provide functions to compute magnitude and phase, and apply magnitude or complex-valued masks. We also provide interfaces to NumPy [25] and torchaudio [7]. Additionally, Griffin-Lim [26] [27] and multi-input spectrogram inversion (MISI) [28] algorithms are provided.

3.2. Masker network

Asteroid provides implementations of widely used masker networks: TaNet’s stacked long short-term memory (LSTM) network [8], Conv-TasNet’s temporal convolutional network (with or without skip connections) [9], and the dual-path recurrent neural network (DPRNN) in [16]. Open-Unmix [13] is also supported for music source separation.

3.3. Loss functions — Permutation invariance

Asteroid supports several loss functions: mean squared error, scale-invariant signal-to-distortion ratio (SI-SDR) [9] [29], scale-dependent SDR [29], signal-to-noise ratio (SNR), perceptual evaluation of speech quality (PESQ) [30], and affinity loss for deep clustering [6].

Whenever the sources are of the same nature, a permutation-invariant (PIT) loss shall be used [7] [31]. Asteroid provides an optimized, versatile implementation of PIT losses. Let \( s = \{s_j(t)\}_{j=1..J} \) and \( \hat{s} = \{\hat{s}_j(t)\}_{j=1..J} \) be the matrices of true and estimated source signals, respectively. We denote as \( \hat{s}_\sigma = \{\hat{s}_{\sigma(j)}(t)\}_{j=1..J} \) a permutation of \( s \) by \( \sigma \in S_J \), where \( S_J \) is the set of permutations of \([1, \ldots, J]\). A PIT loss \( \mathcal{L}_{\text{PIT}} \) is defined as

\[
\mathcal{L}_{\text{PIT}}(\theta) = \min_{\sigma \in S_J} \mathcal{L}(\hat{s}_\sigma, s),
\]

where \( L \) is a classical (permutation-dependent) loss function, which depends on the network’s parameters \( \theta \) through \( \hat{s}_\sigma \).

We assume that, for a given permutation hypothesis \( \sigma \), the loss \( \mathcal{L}(\hat{s}_\sigma, s) \) can be written as

\[
\mathcal{L}(\hat{s}_\sigma, s) = G(\mathcal{F}(\hat{s}_{\sigma(1)}, s_1), \ldots, \mathcal{F}(\hat{s}_{\sigma(J)}, s_J))
\]

where \( s_j = [s_j(0), \ldots, s_j(T)] \), \( \hat{s}_j = [\hat{s}_j(0), \ldots, \hat{s}_j(T)] \), \( \mathcal{F} \) computes the pairwise loss between a single true source and its hypothesized estimate, and \( G \) is the reduce function, usually a simple mean operation. Denoting by \( \mathcal{P} \) the \( J \times J \) pairwise loss matrix with entries \( \mathcal{P}(\hat{s}_j, s_j) \), we can rewrite (2) as

\[
\mathcal{L}_{\text{PIT}}(\theta) = \min_{\sigma \in S_J} G(\mathcal{P}(\hat{s}_{\sigma(1)}, s_1), \ldots, \mathcal{P}(\hat{s}_{\sigma(J)}, s_J))
\]

and reduce the computational complexity from \( J! \) to \( J^2 \) by pre-computing \( \mathcal{P} \)’s terms. Taking advantage of this, Asteroid provides PITLossWrapper, a simple yet powerful class that can efficiently turn any pairwise loss \( \mathcal{P} \) or permutation-dependent loss \( \mathcal{L} \) into a PIT loss.

3.4. Datasets

Asteroid provides baseline recipes for the following datasets: wsj0-2mix and wsj0-3mix [6], WHAM [22], WHAMR [23], LibriMix [24], FUSS [25], Microsoft’s Deep Noise Suppression challenge dataset (DNS) [26], SMS-WSJ [27], Kinect-WSJ [28], and MUSDB18 [29]. Their characteristics are summarized and compared in Table 1. wsj0-2mix and MUSDB18 are today’s reference datasets for speech and music separation, respectively. WHAM, WHAMR, LibriMix, SMS-WSJ and Kinect-WSJ are recently released datasets which address some shortcomings of wsj0-2mix. FUSU is the first open-source dataset to tackle the separation of arbitrary sounds. Note that wsj0-2mix is a subset of WHAM which is a subset of WHAMR.
Table 1: Datasets currently supported by Asteroid. * White sensor noise. ** Background environmental scenes.

<table>
<thead>
<tr>
<th>Source types</th>
<th>wsj0-mix</th>
<th>WHAM</th>
<th>WHAMR</th>
<th>Librimix</th>
<th>DNS</th>
<th>SMS-WSJ</th>
<th>Kinect-WSJ</th>
<th>MUSDB18</th>
<th>FUSS</th>
</tr>
</thead>
<tbody>
<tr>
<td># sources</td>
<td>speech</td>
<td>speech</td>
<td>speech</td>
<td>speech</td>
<td>speech</td>
<td>speech</td>
<td>speech</td>
<td>music</td>
<td>sounds</td>
</tr>
<tr>
<td>Noise</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reverb</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td># channels</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Sampling rate</td>
<td>16k</td>
<td>16k</td>
<td>16k</td>
<td>16k</td>
<td>16k</td>
<td>16k</td>
<td>16k</td>
<td>16k</td>
<td>16k</td>
</tr>
<tr>
<td>Hours</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>210</td>
<td>100</td>
<td>85</td>
<td>30</td>
<td>10</td>
<td>55 (+aug.)</td>
</tr>
</tbody>
</table>

3.5. Training

For training source separation systems, Asteroid offers a thin wrapper around PyTorch-Lightning that seamlessly enables distributed training, experiment logging and more, without sacrificing flexibility. Regarding the optimizers, we also rely on native PyTorch and torch-optimizer. PyTorch provides basic optimizers such as SGD and Adam and torch-optimizer provides state-of-the-art optimizers such as RAdam, Ranger or Yogi.

3.6. Evaluation

Evaluation is performed using \texttt{pb\_bss\_eval}, a sub-toolkit of \texttt{pb\_bss} written for evaluation. It natively supports most metrics used in source separation: SDR, signal-to-interference ratio (SIR), signal-to-artifacts ratio (SAR) \cite{Meyer2015}, SI-SDR \cite{Farid2020}, PESQ \cite{Petersen2005}, and short-time objective intelligibility (STOI) \cite{Barker2012}.

4. Implementation

Asteroid follows Kaldi-style recipes \cite{Povey2011}, which involve several stages as depicted in Fig. 2. These recipes implement the entire pipeline from data download and preparation to model training and evaluation. We show the typical organization of a recipe’s directory in Fig. 3. The entry point of a recipe is the \texttt{run.sh} script which will execute the following stages:

- **Stage 0:** Download data that is needed for the recipe.
- **Stage 1:** Generate mixtures with the official scripts, optionally perform data augmentation.
- **Stage 2:** Gather data information into text files expected by the corresponding \texttt{DataLoader}.
- **Stage 3:** Train the source separation system.
- **Stage 4:** Separate test mixtures and evaluate.

In the first stage, necessary data is downloaded (if available) into a storage directory specified by the user. We use the official scripts provided by the dataset’s authors to generate the data, and optionally perform data augmentation. All the information required by the dataset’s \texttt{DataLoader} such as filenames and paths, utterance lengths, speaker IDs, etc., is then gathered into text files under \texttt{data/}. The training stage is finally followed by the evaluation stage. Throughout the recipe, log files are saved under \texttt{logs/} and generated data is saved under \texttt{exp/}.

As can be seen in Fig. 4, the model class, which is a direct subclass of PyTorch’s \texttt{nn.Module}, is defined in \texttt{model.py}. It is imported in both training and evaluation scripts. Instead of defining constants in \texttt{model.py} and \texttt{train.py}, most of them are gathered in a YAML configuration file \texttt{conf.yml}. An argument parser is created from this configuration file to allow modification of these values from the command line, with \texttt{run.sh} passing arguments to \texttt{train.py}. The resulting modified configuration is saved in \texttt{exp/} to enable future reuse. Other arguments such as the experiment name, the number of GPUs, etc., are directly passed to \texttt{run.sh}.

5. Example results

To illustrate the potential of Asteroid, we compare the performance of state-of-the-art methods as reported in the corresponding papers with our implementation. We do so on two common source separation datasets: wsj0-2mix \cite{Garofolo1993} and WHAMR \cite{Garofalo2019}. The wsj0-2mix consists of a 30 h training set, a 10 h validation set, and a 5 h test set of single-channel two-speaker mixtures without noise and reverberation. Utterances taken from the Wall Street Journal (WSJ) dataset are mixed together at random SNRs between $-5$ dB and $5$ dB. Speakers in the test set are different from those in the training and validation sets. WHAMR
is a noisy and reverberant extension of wsj0-2mix. Experiments are conducted on the 8 kHz min version of both datasets. Note that we use wsj0-2mix separation, WHAM’s clean separation, and WHAMR’s anechoic clean separation tasks interchangeably as the datasets only differ by a global scale.

Table 2 reports SI-SDR improvements (SI-SDR<sub>i</sub>) on the test set of wsj0-2mix for several well-known source separation systems. In Table 3, we reproduce Table 2 from [33] which reports the performance of an improved TasNet architecture (more recurrent units, overlap-add for synthesis) on the four main tasks of WHAMR: anechoic separation, noisy anechoic separation, reverberant separation, and noisy reverberant separation. On all four tasks, Asteroid’s recipes achieved better results than originally reported, by up to 2.6 dB.

In both Tables 2 and 3, we can see that our implementations outperform the original ones in most cases. Most often, the aforementioned architectures are trained on 4-second segments. For the architectures requiring a large amount of memory (e.g., Conv-TasNet and DPRNN), we reduce the length of the training segments in order to increase the batch size and stabilize gradients. This, as well as using a weight decay of $10^{-5}$ for recurrent architectures increased the final performance of our systems.

Asteroid was designed such that writing new code is very simple and results can be quickly obtained. For instance, starting from stage 2, writing the TasNet recipe used in Table 3 took less than a day and the results were simply generated with the command in Fig. 5, where the GPU ID is specified with the --id argument.

Figure 5: Example command line usage.

6. Conclusion

In this paper, we have introduced Asteroid, a new open-source audio source separation toolkit designed for researchers and practitioners. Comparative experiments show that results obtained with Asteroid are competitive on several datasets and for several architectures. The toolkit was designed such that it can quickly be extended with new network architectures or new benchmark datasets. In the near future, pre-trained models will be made available and we intend to interface with ESPNet to enable end-to-end multi-speaker speech recognition.

7. References


