asteroid Documentation

Release 0.6.0

Manuel Pariente et al.

Jun 28, 2022

Start here

Asteroid is a Pytorch-based audio source separation toolkit that enables fast experimentation on common datasets. It comes with a source code that supports a large range of datasets and architectures, and a set of recipes to reproduce some important papers.

CHAPTER 1

What is Asteroid?

Asteroid is a PyTorch-based audio source separation toolkit.

The main goals of Asteroid are:

- Gather a wider **community** around audio source separation by lowering the barriers to entry.
- **Promote reproducibility** by replicating important research papers.
- Automatize most engineering and **make way for research**.
- Simplify **model sharing** to reduce compute costs and carbon footprint.

So, how do we do that? We aim to provide

- PyTorch `Dataset` for **common datasets**.
- Ready-to-use state-of-the art source separation architectures in **native PyTorch**.
- **Configurable recipes** from data preparation to evaluation.
- **Pretrained models** for a wide variety of tasks and architectures.

1.1 Who is it for?

Asteroid has several target usage:

- Use asteroid in your own code, as a package.
- Use available recipes to build your own separation model.
- Use pretrained models to process your files.
- Hit the ground running with your research ideas!

1.2 Want to know more?

- Visit our webpage
- Read our paper
- Watch the presentation video
- Check how we won the PyTorch Hackathon 2020 !

CHAPTER 2

Installation

By following the instructions below, first install PyTorch and then Asteroid (using either pip/dev install). We recommend the development installation for users likely to modify the source code.

2.1 CUDA and PyTorch

Asteroid is based on PyTorch. To run Asteroid on GPU, you will need a CUDA-enabled PyTorch installation. Visit this site for the instructions: https://pytorch.org/get-started/locally/.

2.2 Pip

Asteroid is regularly updated on PyPI, install the latest stable version with:

```
pip install asteroid
```

2.3 Development installation

For development installation, you can fork/clone the GitHub repo and locally install it with pip:

```
git clone https://github.com/asteroid-team/asteroid
cd asteroid
pip install -e .
```

This is an editable install (`-e` flag), it means that source code changes (or branch switching) are automatically taken into account when importing asteroid.

You can also use `conda env create -f environment.yml` to create a Conda env directly.

CHAPTER 3

What is a recipe?

A recipe is a set of scripts that use Asteroid to build a source separation system. Each directory corresponds to a dataset and each subdirectory corresponds to a system build on this dataset. You can start by reading this recipe to get familiar with them.

3.1 How is it organized?

Most recipes are organized as follows. When you clone the repo, `data`, `exp` and `logs` won't be there yet, it's normal.

```
├── data/
├── exp/
├── logs/
├── local/
│   ├── convert_sphere2wav.sh
│   ├── prepare_data.sh
│   ├── conf.yml
│   └── preprocess_wham.py
├── utils/
│   ├── parse_options.sh
│   └── prepare_python_env.sh
├── run.sh
├── train.py
├── model.py
└── eval.py
```

A small graph might help.

3.2 How does it work?

Let's try to summarize how recipes work :

- There is a master file, `run.sh`, from which all the steps are ran (install dependencies, download data, create dataset, train a model evaluate it and so on..). This recipe style is borrowed from Kaldi and ESPnet.
  - You usually have to change some variables in the top of the file (comments are around it to help you) like data directory, python path etc..
  - This script is controlled by several arguments. Among them, `stage` controls from where do you start the script. You already generated the data? No need to do it again, set `stage=3`!
  - All steps until training are dataset-specific and the corresponding scripts are stored in `./local`
- The training and evaluation scripts are then called from `run.sh`
  - There is a script, `model.py`, where the model should be defined along with the `System` subclass used for training (if needed).
  - We wrap the model definition in one function (`make_model_and_optimizer`). The function receives a dictionary which is also saved in the experiment folder. This make checkpoint restoring easy without any additional constraints.
  - We also write a function to load the best model (`load_best_model`) after training. This is useful to load the model several time (evaluation, separation of new examples...).
- The arguments flow through bash/python/yaml in a specific way, which was designed by us and suits our use-cases until now:
  - The very first step is the `local/conf.yml` file where is a hierarchical configuration file,
  - **On the python side** : This file is parsed as a dictionary of dictionary in `training.py`. From this dict, we create an argument parser which can accept all the second-level keys from the dictionary (so second-level keys should be unique) as arguments and has the default values from the `conf.yml` file.
  - **On the bash side**: we also parse arguments from the command line (using `utils/parse_options.sh`). The arguments above the line `. utils/parse_options.sh` can be parsed, the rest are fixed. Most arguments will be passed to the training script. Others control the data preparation, GPU usage etc...
  - In light of all this the config file should have sensible default values that shouldn't be modified by hand much. The quickly configurable part of the recipe are added to `run.sh` (you want to experiment with the batch size, add an argument in and pass it to python. If you want it fixed, no need to put it in bash, the conf.yml file keeps it for you.) This makes it possible to directly identify the important parts of the experiment, without reading lots of lines of argparser or bash arguments.
- Some more notes :
  - After the first execution, you can go and change `stage` in `run.sh` to avoid redoing all the steps every-time.
  - To use GPUs for training, run `run.sh --id 0,1` where `0` and `1` are the GPUs you want to use, training should automatically take advantage of both GPUs.
  - By default, a random id is generated for each run, you can also add a `tag` to name the experiments how you want. For example `run.sh --tag with_cool_loss` will save all results to `exp/train_{arch_name}_with_cool_loss`. You'll also find the corresponding log file in `logs/train_{arch_name}_with_cool_loss.log`.
  - Model loading methods suppose that the model architecture is the same as when training was performed. Be careful when you change it.

Again, you have a doubt, a question, a suggestion or a request, open an issue or join the slack, we'll be happy to help you.

CHAPTER 4

Datasets and tasks

The following is a list of supported datasets, sorted by task. If you're more interested in the corresponding PyTorch `Dataset`, see *this page*

4.1 Speech separation

4.1.1 wsj0-2mix dataset

wsj0-2mix is a single channel speech separation dataset base on WSJ0. Three speaker extension (wsj0-3mix) is also considered here.

Reference

```
@article{Hershey_2016,
   title={Deep clustering: Discriminative embeddings for segmentation and separation},
   ISBN={9781479999880},
   url={http://dx.doi.org/10.1109/ICASSP.2016.7471631},
   DOI={10.1109/icassp.2016.7471631},
   journal={2016 IEEE International Conference on Acoustics, Speech and Signal␣
↪Processing (ICASSP)},
   publisher={IEEE},
   author={Hershey, John R. and Chen, Zhuo and Le Roux, Jonathan and Watanabe, Shinji}
↪,
   year={2016},
}
```

4.1.2 WHAM dataset

WHAM! is a noisy single-channel speech separation dataset based on WSJ0. It is a noisy extension of wsj0-2mix.

More info here.

References

```
@inproceedings{WHAMWichern2019,
  author={Gordon Wichern and Joe Antognini and Michael Flynn and Licheng Richard Zhu␣
↪and Emmett McQuinn and Dwight Crow and Ethan Manilow and Jonathan Le Roux},
  title={{WHAM!: extending speech separation to noisy environments}},
  year=2019,
  booktitle={Proc. Interspeech},
  pages={1368--1372},
  doi={10.21437/Interspeech.2019-2821},
  url={http://dx.doi.org/10.21437/Interspeech.2019-2821}
}
```

4.1.3 WHAMR dataset

WHAMR! is a noisy and reverberant single-channel speech separation dataset based on WSJ0. It is a reverberant extension of WHAM!.

Note that WHAMR! can synthesize binaural recordings, but we only consider the single channel for now.

More info here. **References**

```
@misc{maciejewski2019whamr,
    title={WHAMR!: Noisy and Reverberant Single-Channel Speech Separation},
    author={Matthew Maciejewski and Gordon Wichern and Emmett McQuinn and Jonathan Le␣
↪Roux},
    year={2019},
    eprint={1910.10279},
    archivePrefix={arXiv},
    primaryClass={cs.SD}
}
```

4.1.4 LibriMix dataset

The LibriMix dataset is an open source dataset derived from LibriSpeech dataset. It's meant as an alternative and complement to WHAM.

More info here.

References

```
@misc{cosentino2020librimix,
    title={LibriMix: An Open-Source Dataset for Generalizable Speech Separation},
    author={Joris Cosentino and Manuel Pariente and Samuele Cornell and Antoine␣
↪Deleforge and Emmanuel Vincent},
    year={2020},
    eprint={2005.11262},
    archivePrefix={arXiv},
    primaryClass={eess.AS}
}
```

4.1.5 Kinect-WSJ dataset

Kinect-WSJ is a reverberated, noisy version of the WSJ0-2MIX dataset. Microphones are placed on a linear array with spacing between the devices resembling that of Microsoft Kinect ™, the device used to record the CHiME-5 dataset.

This was done so that we could use the real ambient noise captured as part of CHiME-5 dataset. The room impulse responses (RIR) were simulated for a sampling rate of 16,000 Hz.

Requirements

- wsj_path : Path to precomputed wsj-2mix dataset. Should contain the folder 2speakers/wav16k/. If you don't have wsj_mix dataset, please create it using the scripts in egs/wsj0_mix
- chime_path : Path to chime-5 dataset. Should contain the folders train, dev and eval
- dihard_path : Path to dihard labels. Should contain `*.lab` files for the train and dev set

References Original repo

```
@inproceedings{sivasankaran2020,
  booktitle = {2020 28th {{European Signal Processing Conference}} ({{EUSIPCO}})},
  title={Analyzing the impact of speaker localization errors on speech separation for␣
↪automatic speech recognition},
  author={Sunit Sivasankaran and Emmanuel Vincent and Dominique Fohr},
  year={2021},
  month = Jan,
}
```

4.1.6 SMS_WSJ dataset

SMS_WSJ (stands for Spatialized Multi-Speaker Wall Street Journal) is a multichannel source separation dataset, based on WSJ0 and WSJ1.

All the information regarding the dataset can be found in this repo.

References If you use this dataset, please cite the corresponding paper as follows :

```
@Article{SmsWsj19,
  author    = {Drude, Lukas and Heitkaemper, Jens and Boeddeker, Christoph and Haeb-
↪Umbach, Reinhold},
  title     = {{SMS-WSJ}: Database, performance measures, and baseline recipe for␣
↪multi-channel source separation and recognition},
  journal   = {arXiv preprint arXiv:1910.13934},
  year      = {2019},
}
```

4.2 Speech enhancement

4.2.1 DNS Challenge's dataset

The Deep Noise Suppression (DNS) Challenge is a single-channel speech enhancement challenge organized by Microsoft, with a focus on real-time applications. More info can be found on the official page.

References The challenge paper, here.

```
@misc{DNSChallenge2020,
title={The INTERSPEECH 2020 Deep Noise Suppression Challenge: Datasets, Subjective␣
↪Speech Quality and Testing Framework},
author={Chandan K. A. Reddy and Ebrahim Beyrami and Harishchandra Dubey and Vishak␣
↪Gopal and Roger Cheng and Ross Cutler and Sergiy Matusevych and Robert Aichner and␣
↪Ashkan Aazami and Sebastian Braun and Puneet Rana and Sriram Srinivasan and␣
↪Johannes Gehrke}, year={2020},
```

(continues on next page)

(continued from previous page)

```
eprint={2001.08662},
}
```

The baseline paper, here.

```
@misc{xia2020weighted,
title={Weighted Speech Distortion Losses for Neural-network-based Real-time Speech␣
↪Enhancement},
author={Yangyang Xia and Sebastian Braun and Chandan K. A. Reddy and Harishchandra␣
↪Dubey and Ross Cutler and Ivan Tashev},
year={2020},
eprint={2001.10601},
}
```

4.3 Music source separation

4.3.1 MUSDB18 Dataset

The musdb18 is a dataset of 150 full lengths music tracks (~10h duration) of different genres along with their isolated drums, bass, vocals and others stems.

More info here.

4.3.2 DAMP-VSEP dataset

All the information regarding the dataset can be found in zenodo.

References If you use this dataset, please cite as follows :

```
@dataset{smule_inc_2019_3553059,
  author       = {Smule, Inc},
  title        = {{DAMP-VSEP: Smule Digital Archive of Mobile
                   Performances - Vocal Separation}},
  month        = oct,
  year         = 2019,
  publisher    = {Zenodo},
  version      = {1.0.1},
  doi          = {10.5281/zenodo.3553059},
  url          = {https://doi.org/10.5281/zenodo.3553059}
}
```

4.4 Environmental sound separation

4.4.1 FUSS dataset

The Free Universal Sound Separation (FUSS) dataset comprises audio mixtures of arbitrary sounds with source references for use in experiments on arbitrary sound separation.

All the information related to this dataset can be found in this repo.

References If you use this dataset, please cite the corresponding paper as follows:

```
@Article{Wisdom2020,
  author    = {Scott Wisdom and Hakan Erdogan and Daniel P. W. Ellis and Romain␣
↪Serizel and Nicolas Turpault and Eduardo Fonseca and Justin Salamon and Prem␣
↪Seetharaman and John R. Hershey},
  title     = {What's All the FUSS About Free Universal Sound Separation Data?},
  journal   = {in preparation},
  year      = {2020},
}
```

4.5 Audio-visual source separation

4.5.1 AVSpeech dataset

AVSpeech is an audio-visual speech separation dataset which was introduced by Google in this article Looking to Listen at the Cocktail Party: A Speaker-Independent Audio-Visual Model for Speech Separation.

More info here.

References

```
@article{Ephrat_2018,
   title={Looking to listen at the cocktail party},
   volume={37},
   url={http://dx.doi.org/10.1145/3197517.3201357},
   DOI={10.1145/3197517.3201357},
   journal={ACM Transactions on Graphics},
   publisher={Association for Computing Machinery (ACM)},
   author={Ephrat, Ariel and Mosseri, Inbar and Lang, Oran and Dekel, Tali and Wilson,
↪ Kevin and Hassidim, Avinatan and Freeman, William T. and Rubinstein, Michael},
   year={2018},
   pages={1-11}
}
```

4.6 Speaker extraction

CHAPTER 5

Training and Evaluation

Training and evaluation are the two essential parts of the recipes. For training, we offer a thin wrapper around PyTorchLightning that seamlessly enables distributed training, experiment logging and more, without sacrificing flexibility. For evaluation we released `pb_bss_eval` on PyPI, which is the evaluation part of pb_bss. All the credit goes to the original authors from the Paderborn University.

5.1 Training with PyTorchLightning

First, have a look here for an overview of PyTorchLightning. As you saw, the `LightningModule` is a central class of PyTorchLightning where a large part of the research-related logic lives. Instead of subclassing it everytime, we use `System`, a thin wrapper that separately gathers the essential parts of every deep learning project:

1. A model
2. Optimizer
3. Loss function
4. Train/val data

```
class System(pl.LightningModule):
        def __init__(self, model, optimizer, loss_func, train_loader, val_loader):
                ...

        def common_step(self, batch):
         """ common_step is the method that'll be called at both train/val time. """
                inputs, targets = batch
                est_targets = self(inputs)
                loss = self.loss_func(est_targets, targets)
                return loss
```

Only overwriting `common_step` will often be enough to obtain a desired behavior, while avoiding boilerplate code. Then, we can use the native PyTorchLightning `Trainer` to train the models.

5.2 Evaluation

Asteroid's function `compute_metrics` that calls `pb_bss_eval` is used to compute the following common source separation metrics:

- SDR / SIR / SAR
- STOI
- PESQ
- SI-SDR

CHAPTER 6

Pretrained models

Asteroid provides pretrained models through Hugging Face's Model Hub. Have a look at this page to choose which model you want to use.

Enjoy having pretrained models? **Please share your models** if you train some :pray: It's really simple with the Hub, check the next sections.

6.1 Using them

Loading a pretrained model is super simple!

```
from asteroid.models import ConvTasNet
model = ConvTasNet.from_pretrained('mpariente/ConvTasNet_WHAM!_sepclean')
```

You can also load it with Hub

```
from torch import hub
model = hub.load('mpariente/asteroid', 'conv_tasnet', 'mpariente/ConvTasNet_WHAM!_
↪sepclean')
```

6.2 Model caching

When using a `from_pretrained` method, the model is downloaded and cached. The cache directory is either the value in the `$ASTEROID_CACHE` environment variable, or `~/.cache/torch/asteroid`.

6.3 Share your models

At the end of each sharing-enabled recipe, all the necessary infos are gathered into a file, the only thing that's left to is to add it to the Model Hub. After creating an account (here), you can

- Add a new model here. with a name like `{model_name}_{dataset_name}_{task}_{sampling_rate}`.
- Clone the repo (`git clone the_URL_youre_at`), cd into it.
- Copy the `model_card_template.md` and fill in the missing information.
- Move the pretrained model in the folder, rename it `pytorch.bin`.
- Register files and commit `git add . && git commit -m "Model release: v1"`.
- And push :tada: `git push` :tada:
- Thank you! :pray:

You can have a look at the docs for more details!

6.4 Note about licenses

All Asteroid's pretrained models are shared under the Attribution-ShareAlike 3.0 (CC BY-SA 3.0) license. This means that models are released under the same license as the original training data. **If any non-commercial data is used during training (wsj0, WHAM's noises etc..), the models are non-commercial use only.** This is indicated in the bottom of the model page (ex: here on the bottom).

CHAPTER 7

Command-line interface

7.1 Inference

7.1.1 asteroid-infer

Example

```
asteroid-infer "mpariente/ConvTasNet_WHAM!_sepclean" --files myaudio.wav --resample --
↪ola-window 8000 --ola-hop 4000
```

Reference

Command ['asteroid-infer', '–help'] failed: [Errno 2] No such file or directory: 'asteroid-infer': 'asteroid-infer'

7.2 Publishing models

7.2.1 asteroid-upload

Reference

Command ['asteroid-upload', '–help'] failed: [Errno 2] No such file or directory: 'asteroid-upload': 'asteroid-upload'

7.2.2 asteroid-register-sr

Reference

Command ['asteroid-register-sr', '–help'] failed: [Errno 2] No such file or directory: 'asteroid-register-sr': 'asteroid-register-sr'

CHAPTER 8

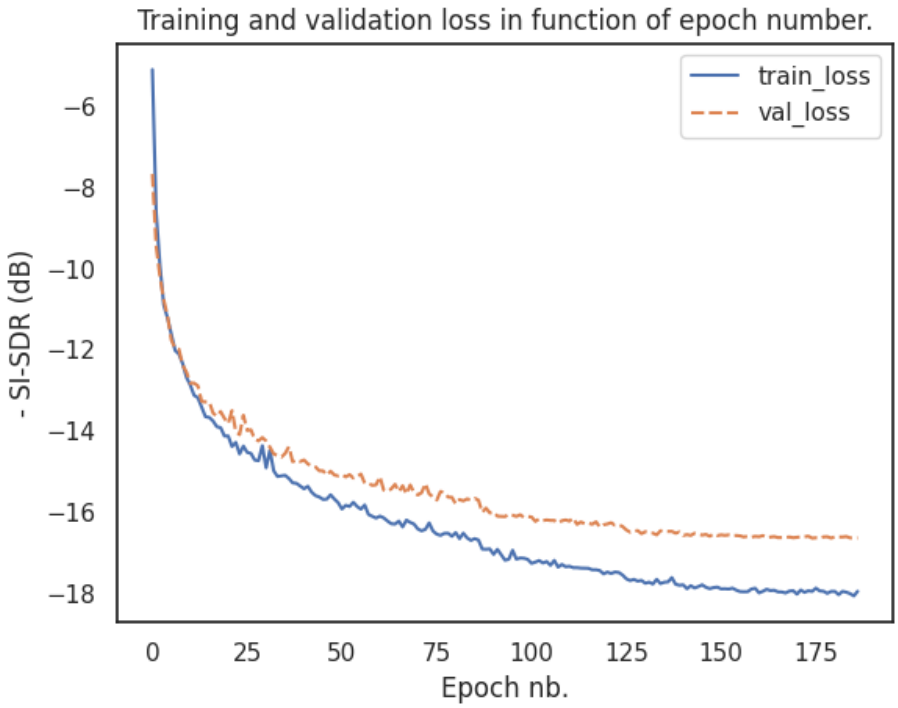
FAQ

8.1 My results are worse than the ones reported in the README, why?

There are few possibilities here:

1. Your data is wrong. We had this examples with wsj0-mix, WHAM etc.. where wv2 was used instead of wv1 to generate the data. This was fixed in #166. Chances are there is a pretrained model available for the given dataset, run the evaluation with it. If your results are different, it's a data problem. Refs: #164, #165 and #188.

2. You stopped training too early. We've seen this happen, specially with DPRNN. Be sure that your training/validation losses are completely flat at the end of training.



3. If it's not both, there is a real bug and we're happy you caught it! Please, open an issue with your torch/pytorch_lightning/asteroid versions to let us know.

8.2 How long does it take to train a model?

Need a log here.

8.3 Can I use the pretrained models for commercial purposes?

Not always. See the note on pretrained models Licenses *Note about licenses*

8.4 Separated audio is really bad, what is happening?

There are several possible cause to this, a common one is clipping.

1. When training with scale invariant losses (e.g. SI-SNR) the audio output can be unbounded. However, waveform values should be normalized to [-1, 1] range before saving, otherwise they will be clipped. See Clipping on Wikipedia and issue #250

2. As all supervised learning approaches, source separation can suffer from generalization error when evaluated on unseen data. If your model works well on data similar to your training data but doesn't work on real data, that's probably why. More about this on Wikipedia.

CHAPTER 9

PyTorch Datasets

This page lists the supported datasets and their corresponding PyTorch's `Dataset` class. If you're interested in the datasets more than in the code, see *this page*.

9.1 LibriMix

class asteroid.data.**LibriMix**(*csv_dir*, *task='sep_clean'*, *sample_rate=16000*, *n_src=2*, *segment=3*, *return_id=False*)

Bases: sphinx.ext.autodoc.importer._MockObject

Dataset class for LibriMix source separation tasks.

Parameters

- **csv_dir** (*str*) – The path to the metadata file.
- **task** (*str*) – One of `'enh_single'`, `'enh_both'`, `'sep_clean'` or `'sep_noisy'`:
  - `'enh_single'` for single speaker speech enhancement.
  - `'enh_both'` for multi speaker speech enhancement.
  - `'sep_clean'` for two-speaker clean source separation.
  - `'sep_noisy'` for two-speaker noisy source separation.
- **sample_rate** (*int*) – The sample rate of the sources and mixtures.
- **n_src** (*int*) – The number of sources in the mixture.
- **segment** (*int, optional*) – The desired sources and mixtures length in s.

References [1] "LibriMix: An Open-Source Dataset for Generalizable Speech Separation", Cosentino et al. 2020.

classmethod loaders_from_mini(*batch_size=4*, ***kwargs*)
Downloads MiniLibriMix and returns train and validation DataLoader.

Parameters

- **batch_size** (*int*) – Batch size of the Dataloader. Only DataLoader param. To have more control on Dataloader, call *mini_from_download* and instantiate the DatalLoader.
- ****kwargs** – keyword arguments to pass the *LibriMix*, see *__init__*. The kwargs will be fed to both the training set and validation set.

Returns *train_loader, val_loader* – training and validation DataLoader out of *LibriMix* Dataset.

Examples

```
>>> from asteroid.data import LibriMix
>>> train_loader, val_loader = LibriMix.loaders_from_mini(
>>>     task='sep_clean', batch_size=4
>>> )
```

classmethod mini_from_download(***kwargs*)
Downloads MiniLibriMix and returns train and validation Dataset. If you want to instantiate the Dataset by yourself, call *mini_download* that returns the path to the path to the metadata files.

Parameters ****kwargs** – keyword arguments to pass the *LibriMix*, see *__init__*. The kwargs will be fed to both the training set and validation set

Returns *train_set, val_set* – training and validation instances of *LibriMix* (data.Dataset).

Examples

```
>>> from asteroid.data import LibriMix
>>> train_set, val_set = LibriMix.mini_from_download(task='sep_clean')
```

static mini_download()
Downloads MiniLibriMix from Zenodo in current directory

Returns The path to the metadata directory.

get_infos()
Get dataset infos (for publishing models).

Returns dict, dataset infos with keys *dataset*, *task* and *licences*.

9.2 Wsj0mix

class asteroid.data.**Wsj0mixDataset**(*json_dir*, *n_src=2*, *sample_rate=8000*, *segment=4.0*)
Bases: sphinx.ext.autodoc.importer._MockObject

Dataset class for the wsj0-mix source separation dataset.

Parameters

- **json_dir** (*str*) – The path to the directory containing the json files.
- **sample_rate** (*int, optional*) – The sampling rate of the wav files.
- **segment** (*float, optional*) – Length of the segments used for training, in seconds. If None, use full utterances (e.g. for test).

- **n_src** (*int, optional*) – Number of sources in the training targets.

References "Deep clustering: Discriminative embeddings for segmentation and separation", Hershey et al. 2015.

__getitem__(*idx*)
Gets a mixture/sources pair. :returns: mixture, vstack([source_arrays])

get_infos()
Get dataset infos (for publishing models).

Returns dict, dataset infos with keys *dataset*, *task* and *licences*.

9.3 WHAM!

class asteroid.data.**WhamDataset**(*json_dir*, *task*, *sample_rate=8000*, *segment=4.0*, *nondefault_nsrc=None*, *normalize_audio=False*)
Bases: sphinx.ext.autodoc.importer._MockObject

Dataset class for WHAM source separation and speech enhancement tasks.

Parameters

- **json_dir** (*str*) – The path to the directory containing the json files.
- **task** (*str*) – One of `'enh_single'`, `'enh_both'`, `'sep_clean'` or `'sep_noisy'`.
  - `'enh_single'` for single speaker speech enhancement.
  - `'enh_both'` for multi speaker speech enhancement.
  - `'sep_clean'` for two-speaker clean source separation.
  - `'sep_noisy'` for two-speaker noisy source separation.
- **sample_rate** (*int, optional*) – The sampling rate of the wav files.
- **segment** (*float, optional*) – Length of the segments used for training, in seconds. If None, use full utterances (e.g. for test).
- **nondefault_nsrc** (*int, optional*) – Number of sources in the training targets. If None, defaults to one for enhancement tasks and two for separation tasks.
- **normalize_audio** (*bool*) – If True then both sources and the mixture are normalized with the standard deviation of the mixture.

References "WHAM!: Extending Speech Separation to Noisy Environments", Wichern et al. 2019

__getitem__(*idx*)
Gets a mixture/sources pair. :returns: mixture, vstack([source_arrays])

get_infos()
Get dataset infos (for publishing models).

Returns dict, dataset infos with keys *dataset*, *task* and *licences*.

9.4 WHAMR!

class asteroid.data.**WhamRDataset**(*json_dir*, *task*, *sample_rate=8000*, *segment=4.0*, *nondefault_nsrc=None*)

Bases: sphinx.ext.autodoc.importer._MockObject

Dataset class for WHAMR source separation and speech enhancement tasks.

Parameters

- **json_dir** (*str*) – The path to the directory containing the json files.
- **task** (*str*) – One of `'sep_clean'`, `'sep_noisy'`, `'sep_reverb'` or `'sep_reverb_noisy'`.
  - `'sep_clean'` for two-speaker clean (anechoic) source separation.
  - `'sep_noisy'` for two-speaker noisy (anechoic) source separation.
  - `'sep_reverb'` for two-speaker clean reverberant source separation.
  - `'sep_reverb_noisy'` for two-speaker noisy reverberant source separation.
- **sample_rate** (*int, optional*) – The sampling rate of the wav files.
- **segment** (*float, optional*) – Length of the segments used for training, in seconds. If None, use full utterances (e.g. for test).
- **nondefault_nsrc** (*int, optional*) – Number of sources in the training targets. If None, defaults to one for enhancement tasks and two for separation tasks.

References "WHAMR!: Noisy and Reverberant Single-Channel Speech Separation", Maciejewski et al. 2020

__getitem__(*idx*)

Gets a mixture/sources pair. :returns: mixture, vstack([source_arrays])

get_infos()

Get dataset infos (for publishing models).

Returns dict, dataset infos with keys *dataset*, *task* and *licences*.

9.5 SMS-WSJ

class asteroid.data.**SmsWsjDataset**(*json_path*, *target*, *dset*, *sample_rate=8000*, *single_channel=True*, *segment=4.0*, *nondefault_nsrc=None*, *normalize_audio=False*)

Bases: sphinx.ext.autodoc.importer._MockObject

Dataset class for SMS WSJ source separation.

Parameters

- **json_path** (*str*) – The path to the sms_wsj json file.
- **target** (*str*) – One of `'source'`, `'early'` or `'image'`.
  - `'source'` non reverberant clean targets signals.
  - `'early'` early reverberation target signals.
  - `'image'` reverberant target signals
- **dset** (*str*) – train_si284 for train, cv_dev93 for validation and test_eval92 for test

- **sample_rate** (*int, optional*) – The sampling rate of the wav files.
- **segment** (*float, optional*) – Length of the segments used for training, in seconds. If None, use full utterances (e.g. for test).
- **single_channel** (*bool*) – if False all channels are used if True only a random channel is used during training and the first channel during test
- **nondefault_nsrc** (*int, optional*) – Number of sources in the training targets.
- **normalize_audio** (*bool*) – If True then both sources and the mixture are normalized with the standard deviation of the mixture.

References "SMS-WSJ: Database, performance measures, and baseline recipe for multi-channel source separation and recognition", Drude et al. 2019

__getitem__(*idx*)
Gets a mixture/sources pair. :returns: mixture, vstack([source_arrays])

get_infos()
Get dataset infos (for publishing models).

Returns dict, dataset infos with keys *dataset*, *task* and *target*.

9.6 KinectWSJMix

class asteroid.data.**KinectWsjMixDataset**(*json_dir*, *n_src=2*, *sample_rate=16000*, *segment=4.0*)
Bases: asteroid.data.wsj0_mix.Wsj0mixDataset

Dataset class for the KinectWSJ-mix source separation dataset.

Parameters

- **json_dir** (*str*) – The path to the directory containing the json files.
- **sample_rate** (*int, optional*) – The sampling rate of the wav files.
- **segment** (*float, optional*) – Length of the segments used for training, in seconds. If None, use full utterances (e.g. for test).
- **n_src** (*int, optional*) – Number of sources in the training targets.

References "Analyzing the impact of speaker localization errors on speech separation for automatic speech recognition", Sunit Sivasankaran et al. 2020.

__getitem__(*idx*)
Gets a mixture/sources pair. :returns: mixture, stack([source_arrays]), noise

mixture is of dimension [samples, channels] sources are of dimension [n_src, samples, channels]

get_infos()
Get dataset infos (for publishing models).

Returns dict, dataset infos with keys *dataset*, *task* and *licences*.

9.7 DNSDataset

class asteroid.data.**DNSDataset**(*json_dir*)
Bases: sphinx.ext.autodoc.importer._MockObject

Deep Noise Suppression (DNS) Challenge's dataset.

Args json_dir (str): path to the JSON directory (from the recipe).

References "The INTERSPEECH 2020 Deep Noise Suppression Challenge: Datasets, Subjective Testing Framework, and Challenge Results", Reddy et al. 2020.

__getitem__(*idx*)
Gets a mixture/sources pair. :returns: mixture, vstack([source_arrays])

get_infos()
Get dataset infos (for publishing models).

Returns dict, dataset infos with keys *dataset*, *task* and *licences*.

9.8 MUSDB18

class asteroid.data.**MUSDB18Dataset**(*root, sources=['vocals', 'bass', 'drums', 'other'], targets=None, suffix='.wav', split='train', subset=None, segment=None, samples_per_track=1, random_segments=False, random_track_mix=False, source_augmentations=<function MUSDB18Dataset.<lambda>>, sample_rate=44100*)
Bases: sphinx.ext.autodoc.importer._MockObject

MUSDB18 music separation dataset

The dataset consists of 150 full lengths music tracks (~10h duration) of different genres along with their isolated stems:

drums, *bass*, *vocals* and *others*.

Out-of-the-box, asteroid does only support MUSDB18-HQ which comes as uncompressed WAV files. To use the MUSDB18, please convert it to WAV first:

- MUSDB18 HQ: https://zenodo.org/record/3338373
- MUSDB18 https://zenodo.org/record/1117372

Note: The datasets are hosted on Zenodo and require that users request access, since the tracks can only be used for academic purposes. We manually check this requests.

This dataset asssumes music tracks in (sub)folders where each folder has a fixed number of sources (defaults to 4). For each track, a list of *sources* and a common *suffix* can be specified. A linear mix is performed on the fly by summing up the sources

Due to the fact that all tracks comprise the exact same set of sources, random track mixing can be used can be used, where sources from different tracks are mixed together.

Folder Structure:

```
>>> #train/1/vocals.wav ---------|
>>> #train/1/drums.wav ----------+--> input (mix), output[target]
>>> #train/1/bass.wav -----------|
>>> #train/1/other.wav ---------/
```

Parameters

- **root** (*str*) – Root path of dataset
- **sources** (list of str, optional) – List of source names that composes the mixture. Defaults to MUSDB18 4 stem scenario: *vocals*, *drums*, *bass*, *other*.
- **targets** (*list or None, optional*) – List of source names to be used as targets. If None, a dict with the 4 stems is returned.

  If e.g [*vocals*, *drums*], a tensor with stacked *vocals* and *drums* is returned instead of a dict. Defaults to None.
- **suffix** (*str, optional*) – Filename suffix, defaults to *.wav*.
- **split** (*str, optional*) – Dataset subfolder, defaults to *train*.
- **subset** (list of str, optional) – Selects a specific of list of tracks to be loaded, defaults to *None* (loads all tracks).
- **segment** (*float, optional*) – Duration of segments in seconds, defaults to None which loads the full-length audio tracks.
- **samples_per_track** (*int, optional*) – Number of samples yielded from each track, can be used to increase dataset size, defaults to *1*.
- **random_segments** (*boolean, optional*) – Enables random offset for track segments.
- **boolean** (*random_track_mix*) – enables mixing of random sources from different tracks to assemble mix.
- **source_augmentations** (list of callable) – list of augmentation function names, defaults to no-op augmentations (input = output)
- **sample_rate** (*int, optional*) – Samplerate of files in dataset.

Variables

- **root** (*str*) – Root path of dataset
- **sources** (list of str, optional) – List of source names. Defaults to MUSDB18 4 stem scenario: *vocals*, *drums*, *bass*, *other*.
- **suffix** (*str, optional*) – Filename suffix, defaults to *.wav*.
- **split** (*str, optional*) – Dataset subfolder, defaults to *train*.
- **subset** (list of str, optional) – Selects a specific of list of tracks to be loaded, defaults to *None* (loads all tracks).
- **segment** (*float, optional*) – Duration of segments in seconds, defaults to None which loads the full-length audio tracks.
- **samples_per_track** (*int, optional*) – Number of samples yielded from each track, can be used to increase dataset size, defaults to *1*.
- **random_segments** (*boolean, optional*) – Enables random offset for track segments.

- **boolean** (*random_track_mix*) – enables mixing of random sources from different tracks to assemble mix.
- **source_augmentations** (list of callable) – list of augmentation function names, defaults to no-op augmentations (input = output)
- ***sample_rate*** (*int, optional*) – Samplerate of files in dataset.
- **tracks** (list of Dict) – List of track metadata

References "The 2018 Signal Separation Evaluation Campaign" Stoter et al. 2018.

get_tracks()
Loads input and output tracks

get_infos()
Get dataset infos (for publishing models).

Returns dict, dataset infos with keys *dataset*, *task* and *licences*.

9.9 DAMP-VSEP

class asteroid.data.**DAMPVSEPSinglesDataset**(*root_path*, *task*, *split='train_singles'*, *ex_per_track=1*, *random_segments=False*, *sample_rate=16000*, *segment=None*, *norm=None*, *source_augmentations=None*, *mixture='original'*)

Bases: sphinx.ext.autodoc.importer._MockObject

DAMP-VSEP vocal separation dataset

This dataset utilises one of the two preprocessed versions of DAMP-VSEP from https://github.com/groadabike/DAMP-VSEP-Singles aimed for SINGLE SINGER separation.

The DAMP-VSEP dataset is hosted on Zenodo. https://zenodo.org/record/3553059

Parameters

- **root_path** (*str*) – Root path to DAMP-VSEP dataset.
- **task** (*str*) – one of 'enh_vocal',‘‘'separation'‘‘.
  - 'enh_vocal' for vocal enhanced.
  - 'separation' for vocal and background separation.
- **split** (*str*) – one of 'train_english', 'train_singles', 'valid' and 'test'. Default to 'train_singles'.
- **ex_per_track** (*int, optional*) – Number of samples yielded from each track, can be used to increase dataset size, defaults to 1.
- **random_segments** (*boolean, optional*) – Enables random offset for track segments.
- **sample_rate** (*int, optional*) – Sample rate of files in dataset. Default 16000 Hz
- **segment** (*float, optional*) – Duration of segments in seconds, Defaults to None which loads the full-length audio tracks.
- **norm** (*Str, optional*) – Type of normalisation to use. Default to None

  - `'song_level'` use mixture mean and std.
  - `` `None` `` no normalisation
- **source_augmentations** (*Callable, optional*) – Augmentations applied to the sources (only). Default to `None`.
- **mixture** (*str, optional*) – Whether to use the original mixture with non-linear effects or remix sources. Default to original.
  - `'remix'` for use addition to remix the sources.
  - `'original'` for use the original mixture.

Note: There are 2 train set available:

- train_english: Uses all English spoken song. Duets are converted into 2 singles. Totalling 9243 performances and 77Hrs.
- train_singles: Uses all singles performances, discarding all duets. Totalling 20660 performances and 149 hrs.

---

get_tracks()
Loads metadata with tracks info. Raises error if metadata doesn't exist.

get_infos()
Get dataset infos (for publishing models).

Returns dict, dataset infos with keys *dataset*, *task* and *licences*.

9.10 FUSS

class asteroid.data.**FUSSDataset**(*file_list_path*, *return_bg=False*)
Bases: `sphinx.ext.autodoc.importer._MockObject`

Dataset class for FUSS [1] tasks.

Parameters

- **file_list_path** (*str*) – Path to the txt (csv) file created at stage 2 of the recipe.
- **return_bg** (*bool*) – Whether to return the background along the mixture and sources (useful for SIR, SAR computation). Default: False.

References [1] Scott Wisdom et al. "What's All the FUSS About Free Universal Sound Separation Data?", 2020, in preparation.

get_infos()
Get dataset infos (for publishing models).

Returns dict, dataset infos with keys *dataset*, *task* and *licences*.

9.11 AVSpeech

class asteroid.data.**AVSpeechDataset**(*input_df_path: Union[str, pathlib.Path], embed_dir: Union[str, pathlib.Path], n_src=2*)
Bases: `sphinx.ext.autodoc.importer._MockObject`

Audio Visual Speech Separation dataset as described in [1].

Parameters

- **input_df_path** (*str, Path*) – path for combination dataset.
- **embed_dir** (*str, Path*) – path where embeddings are stored.
- **n_src** (*int*) – number of sources.

References [1] "Looking to Listen at the Cocktail Party: A Speaker-Independent Audio-Visual Model for Speech Separation" Ephrat et. al https://arxiv.org/abs/1804.03619

CHAPTER 10

Filterbank API

10.1 Filterbank, Encoder and Decoder

class asteroid_filterbanks.**Filterbank**(*n_filters*, *kernel_size*, *stride=None*, *sample_rate=8000.0*)

Bases: sphinx.ext.autodoc.importer._MockObject

Base Filterbank class. Each subclass has to implement a `filters` method.

Parameters

- **n_filters** (*int*) – Number of filters.
- **kernel_size** (*int*) – Length of the filters.
- **stride** (*int, optional*) – Stride of the conv or transposed conv. (Hop size). If None (default), set to `kernel_size // 2`.
- **sample_rate** (*float*) – Sample rate of the expected audio. Defaults to 8000.

Variables **n_feats_out** (*int*) – Number of output filters.

filters()
Abstract method for filters.

pre_analysis(*wav: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03d10a90>*)
Apply transform before encoder convolution.

post_analysis(*spec: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03d10ad0>*)
Apply transform to encoder convolution.

pre_synthesis(*spec: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03d10b10>*)
Apply transform before decoder transposed convolution.

post_synthesis(*wav: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03d10b50>*)
Apply transform after decoder transposed convolution.

get_config()
Returns dictionary of arguments to re-instantiate the class. Needs to be subclassed if the filterbanks takes additional arguments than `n_filters` `kernel_size` `stride` and `sample_rate`.

class asteroid_filterbanks.**Encoder**(*filterbank*, *is_pinv=False*, *as_conv1d=True*, *padding=0*)
Bases: asteroid_filterbanks.enc_dec._EncDec

Encoder class.

Add encoding methods to Filterbank classes. Not intended to be subclassed.

Parameters

- **filterbank** (*Filterbank*) – The filterbank to use as an encoder.
- **is_pinv** (*bool*) – Whether to be the pseudo inverse of filterbank.
- **as_conv1d** (*bool*) – Whether to behave like nn.Conv1d. If True (default), forwarding input with shape $(batch, 1, time)$ will output a tensor of shape $(batch, freq, conv\_time)$. If False, will output a tensor of shape $(batch, 1, freq, conv\_time)$.
- **padding** (*int*) – Zero-padding added to both sides of the input.

classmethod pinv_of(*filterbank*, ***kwargs*)
Returns an *Encoder*, pseudo inverse of a *Filterbank* or *Decoder*.

forward(*waveform*)
Convolve input waveform with the filters from a filterbank.

Parameters **waveform** (torch.Tensor) – any tensor with samples along the last dimension. The waveform representation with and batch/channel etc.. dimension.

Returns torch.Tensor – The corresponding TF domain signal.

Shapes

```
>>> (time, ) -> (freq, conv_time)
>>> (batch, time) -> (batch, freq, conv_time)  # Avoid
>>> if as_conv1d:
>>>     (batch, 1, time) -> (batch, freq, conv_time)
>>>     (batch, chan, time) -> (batch, chan, freq, conv_time)
>>> else:
>>>     (batch, chan, time) -> (batch, chan, freq, conv_time)
>>> (batch, any, dim, time) -> (batch, any, dim, freq, conv_time)
```

class asteroid_filterbanks.**Decoder**(*filterbank*, *is_pinv=False*, *padding=0*, *output_padding=0*)
Bases: asteroid_filterbanks.enc_dec._EncDec

Decoder class.

Add decoding methods to Filterbank classes. Not intended to be subclassed.

Parameters

- **filterbank** (*Filterbank*) – The filterbank to use as an decoder.
- **is_pinv** (*bool*) – Whether to be the pseudo inverse of filterbank.
- **padding** (*int*) – Zero-padding added to both sides of the input.
- **output_padding** (*int*) – Additional size added to one side of the output shape.

Note: `padding` and `output_padding` arguments are directly passed to `F.conv_transpose1d`.

classmethod pinv_of(*filterbank*)
Returns an Decoder, pseudo inverse of a filterbank or Encoder.

forward(*spec*, *length: Optional[int] = None*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03d19110>
Applies transposed convolution to a TF representation.

This is equivalent to overlap-add.

Parameters

- **spec** (`torch.Tensor`) – 3D or 4D Tensor. The TF representation. (Output of *Encoder.forward()*).
- **length** – desired output length.

Returns `torch.Tensor` – The corresponding time domain signal.

class asteroid_filterbanks.**make_enc_dec**
Creates congruent encoder and decoder from the same filterbank family.

Parameters

- **fb_name** (*str, className*) – Filterbank family from which to make encoder and decoder. To choose among `['free', 'analytic_free', 'param_sinc', 'stft']`. Can also be a class defined in a submodule in this subpackage (e.g. *FreeFB*).
- **n_filters** (*int*) – Number of filters.
- **kernel_size** (*int*) – Length of the filters.
- **stride** (*int, optional*) – Stride of the convolution. If None (default), set to `kernel_size // 2`.
- **sample_rate** (*float*) – Sample rate of the expected audio. Defaults to 8000.0.
- **who_is_pinv** (*str, optional*) – If *None*, no pseudo-inverse filters will be used. If string (among `['encoder', 'decoder']`), decides which of `Encoder` or `Decoder` will be the pseudo inverse of the other one.
- **padding** (*int*) – Zero-padding added to both sides of the input. Passed to Encoder and Decoder.
- **output_padding** (*int*) – Additional size added to one side of the output shape. Passed to Decoder.
- ****kwargs** – Arguments which will be passed to the filterbank class additionally to the usual *n_filters*, *kernel_size* and *stride*. Depends on the filterbank family.

Returns *Encoder*, *Decoder*

class asteroid_filterbanks.**get**
Returns a filterbank class from a string. Returns its input if it is callable (already a *Filterbank* for example).

Parameters **identifier** (*str or Callable or None*) – the filterbank identifier.

Returns *Filterbank* or None

10.2 Learnable filterbanks

10.2.1 Free

class asteroid_filterbanks.free_fb.**FreeFB**(*n_filters*, *kernel_size*, *stride=None*, *sample_rate=8000.0*, ***kwargs*)

Bases: asteroid_filterbanks.enc_dec.Filterbank

Free filterbank without any constraints. Equivalent to nn.Conv1d.

Parameters

- **n_filters** (*int*) – Number of filters.
- **kernel_size** (*int*) – Length of the filters.
- **stride** (*int, optional*) – Stride of the convolution. If None (default), set to kernel_size // 2.
- **sample_rate** (*float*) – Sample rate of the expected audio. Defaults to 8000.

Variables **n_feats_out** (*int*) – Number of output filters.

References [1] : "Filterbank design for end-to-end speech separation". ICASSP 2020. Manuel Pariente, Samuele Cornell, Antoine Deleforge, Emmanuel Vincent.

filters()

Abstract method for filters.

10.2.2 Analytic Free

class asteroid_filterbanks.analytic_free_fb.**AnalyticFreeFB**(*n_filters*, *kernel_size*, *stride=None*, *sample_rate=8000.0*, ***kwargs*)

Bases: asteroid_filterbanks.enc_dec.Filterbank

Free analytic (fully learned with analycity constraints) filterbank. For more details, see [1].

Parameters

- **n_filters** (*int*) – Number of filters. Half of *n_filters* will have parameters, the other half will be the hilbert transforms. *n_filters* should be even.
- **kernel_size** (*int*) – Length of the filters.
- **stride** (*int, optional*) – Stride of the convolution. If None (default), set to kernel_size // 2.
- **sample_rate** (*float*) – Sample rate of the expected audio. Defaults to 8000.

Variables **n_feats_out** (*int*) – Number of output filters.

References [1] : "Filterbank design for end-to-end speech separation". ICASSP 2020. Manuel Pariente, Samuele Cornell, Antoine Deleforge, Emmanuel Vincent.

filters()

Abstract method for filters.

10.2.3 Parameterized Sinc

class asteroid_filterbanks.param_sinc_fb.**ParamSincFB**(*n_filters*, *kernel_size*, *stride=None*, *sample_rate=16000.0*, *min_low_hz=50*, *min_band_hz=50*, ***kwargs*)

Bases: asteroid_filterbanks.enc_dec.Filterbank

Extension of the parameterized filterbank from [1] proposed in [2]. Modified and extended from from https://github.com/mravanelli/SincNet

Parameters

- **n_filters** (*int*) – Number of filters. Half of *n_filters* (the real parts) will have parameters, the other half will correspond to the imaginary parts. *n_filters* should be even.
- **kernel_size** (*int*) – Length of the filters.
- **stride** (*int, optional*) – Stride of the convolution. If None (default), set to `kernel_size // 2`.
- **sample_rate** (*float, optional*) – The sample rate (used for initialization).
- **min_low_hz** (*int, optional*) – Lowest low frequency allowed (Hz).
- **min_band_hz** (*int, optional*) – Lowest band frequency allowed (Hz).

Variables **n_feats_out** (*int*) – Number of output filters.

References [1] : "Speaker Recognition from raw waveform with SincNet". SLT 2018. Mirco Ravanelli, Yoshua Bengio. https://arxiv.org/abs/1808.00158

[2] : "Filterbank design for end-to-end speech separation". ICASSP 2020. Manuel Pariente, Samuele Cornell, Antoine Deleforge, Emmanuel Vincent. https://arxiv.org/abs/1910.10400

filters()
Compute filters from parameters

get_config()
Returns dictionary of arguments to re-instantiate the class.

10.3 Fixed filterbanks

10.3.1 STFT

class asteroid_filterbanks.stft_fb.**STFTFB**(*n_filters*, *kernel_size*, *stride=None*, *window=None*, *sample_rate=8000.0*, ***kwargs*)

Bases: asteroid_filterbanks.enc_dec.Filterbank

STFT filterbank.

Parameters

- **n_filters** (*int*) – Number of filters. Determines the length of the STFT filters before windowing.
- **kernel_size** (*int*) – Length of the filters (i.e the window).

- **stride** (*int, optional*) – Stride of the convolution (hop size). If None (default), set to `kernel_size // 2`.
- **window** (numpy.ndarray, optional) – If None, defaults to `np.sqrt(np.hanning())`.
- **sample_rate** (*float*) – Sample rate of the expected audio. Defaults to 8000.

Variables **n_feats_out** (*int*) – Number of output filters.

filters()
Abstract method for filters.

asteroid_filterbanks.stft_fb.**perfect_synthesis_window**(*analysis_window*, *hop_size*)

Computes a window for perfect synthesis given an analysis window and a hop size.

Parameters

- **analysis_window** (*np.array*) – Analysis window of the transform.
- **hop_size** (*int*) – Hop size in number of samples.

Returns *np.array* – the synthesis window to use for perfectly inverting the STFT.

10.3.2 MelGram

class asteroid_filterbanks.melgram_fb.**MelGramFB**(*n_filters*, *kernel_size*, *stride=None*, *window=None*, *sample_rate=8000.0*, *n_mels=40*, *fmin=0.0*, *fmax=None*, *norm='slaney'*, ***kwargs*)

Bases: *asteroid_filterbanks.stft_fb.STFTFB*

Mel magnitude spectrogram filterbank.

Parameters

- **n_filters** (*int*) – Number of filters. Determines the length of the STFT filters before windowing.
- **kernel_size** (*int*) – Length of the filters (i.e the window).
- **stride** (*int, optional*) – Stride of the convolution (hop size). If None (default), set to `kernel_size // 2`.
- **window** (numpy.ndarray, optional) – If None, defaults to `np.sqrt(np.hanning())`.
- **sample_rate** (*float*) – Sample rate of the expected audio. Defaults to 8000.
- **n_mels** (*int*) – Number of mel bands.
- **fmin** (*float*) – Minimum frequency of the mel filters.
- **fmax** (*float*) – Maximum frequency of the mel filters. Defaults to sample_rate//2.
- **norm** (*str*) – Mel normalization {None, 'slaney', or number}. See *librosa.filters.mel*
- ****kwargs** –

post_analysis(*spec: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03cb63d0>*)
Apply transform to encoder convolution.

get_config()
Returns dictionary of arguments to re-instantiate the class. Needs to be subclassed if the filterbanks takes additional arguments than `n_filters` `kernel_size` `stride` and `sample_rate`.

class asteroid_filterbanks.melgram_fb.**MelScale**(*n_filters*, *sample_rate=8000.0*, *n_mels=40*, *fmin=0.0*, *fmax=None*, *norm='slaney'*)
Bases: sphinx.ext.autodoc.importer._MockObject

Mel-scale filterbank matrix.

Parameters

- **n_filters** (*int*) – Number of filters. Determines the length of the STFT filters before windowing.
- **sample_rate** (*float*) – Sample rate of the expected audio. Defaults to 8000.
- **n_mels** (*int*) – Number of mel bands.
- **fmin** (*float*) – Minimum frequency of the mel filters.
- **fmax** (*float*) – Maximum frequency of the mel filters. Defaults to sample_rate//2.
- **norm** (*str*) – Mel normalization {None, 'slaney', or number}. See *librosa.filters.mel*

10.3.3 MPGT

class asteroid_filterbanks.multiphase_gammatone_fb.**MultiphaseGammatoneFB**(*n_filters=128*, *kernel_size=16*, *sample_rate=8000.0*, *stride=None*, ***kwargs*)
Bases: asteroid_filterbanks.enc_dec.Filterbank

Multi-Phase Gammatone Filterbank as described in [1].

Please cite [1] whenever using this.

Original code repository:

Parameters

- **n_filters** (*int*) – Number of filters.
- **kernel_size** (*int*) – Length of the filters.
- **sample_rate** (*float, optional*) – The sample rate (used for initialization).
- **stride** (*int, optional*) – Stride of the convolution. If None (default), set to `kernel_size // 2`.

References [1] David Ditter, Timo Gerkmann, "A Multi-Phase Gammatone Filterbank for Speech Separation via TasNet", ICASSP 2020 Available: https://ieeexplore.ieee.org/document/9053602/

filters()
Abstract method for filters.

10.4 Transforms

10.4.1 Griffin-Lim and MISI

asteroid_filterbanks.griffin_lim.**griffin_lim**(*mag_specgram*, *stft_enc*, *angles=None*, *istft_dec=None*, *n_iter=6*, *momentum=0.9*)

Estimates matching phase from magnitude spectogram using the 'fast' Griffin Lim algorithm [1].

Parameters

- **mag_specgram** (*torch.Tensor*) – (any, dim, ension, freq, frames) as returned by *Encoder(STFTFB)*, the magnitude spectrogram to be inverted.
- **stft_enc** (*Encoder[STFTFB]*) – The *Encoder(STFTFB())* object that was used to compute the input *mag_spec*.
- **angles** (*None or Tensor*) – Angles to use to initialize the algorithm. If None (default), angles are init with uniform ditribution.
- **istft_dec** (*None or Decoder[STFTFB]*) – Optional Decoder to use to get back to the time domain. If None (default), a perfect reconstruction Decoder is built from *stft_enc*.
- **n_iter** (*int*) – Number of griffin-lim iterations to run.
- **momentum** (*float*) – The momentum of fast Griffin-Lim. Original Griffin-Lim is obtained for momentum=0.

Returns *torch.Tensor* – estimated waveforms of shape (any, dim, ension, time).

Examples

```
>>> stft = Encoder(STFTFB(n_filters=256, kernel_size=256, stride=128))
>>> wav = torch.randn(2, 1, 8000)
>>> spec = stft(wav)
>>> masked_spec = spec * torch.sigmoid(torch.randn_like(spec))
>>> mag = transforms.mag(masked_spec, -2)
>>> est_wav = griffin_lim(mag, stft, n_iter=32)
```

References [1] Perraudin et al. "A fast Griffin-Lim algorithm," WASPAA 2013.

[2] D. W. Griffin and J. S. Lim: "Signal estimation from modified short-time Fourier transform," ASSP 1984.

asteroid_filterbanks.griffin_lim.**misi**(*mixture_wav*, *mag_specgrams*, *stft_enc*, *angles=None*, *istft_dec=None*, *n_iter=6*, *momentum=0.0*, *src_weights=None*, *dim=1*)

Jointly estimates matching phase from magnitude spectograms using the Multiple Input Spectrogram Inversion (MISI) algorithm [1].

Parameters

- **mixture_wav** (*torch.Tensor*) – (batch, time)
- **mag_specgrams** (*torch.Tensor*) – (batch, n_src, freq, frames) as returned by *Encoder(STFTFB)*, the magnitude spectrograms to be jointly inverted using MISI (modified or not).
- **stft_enc** (*Encoder[STFTFB]*) – The *Encoder(STFTFB())* object that was used to compute the input *mag_spec*.

- **angles** (*None or Tensor*) – Angles to use to initialize the algorithm. If None (default), angles are init with uniform ditribution.
- **istft_dec** (*None or Decoder[STFTFB]*) – Optional Decoder to use to get back to the time domain. If None (default), a perfect reconstruction Decoder is built from *stft_enc*.
- **n_iter** (*int*) – Number of MISI iterations to run.
- **momentum** (*float*) – Momentum on updates (this argument comes from GriffinLim). Defaults to 0 as it was never proposed anywhere.
- **src_weights** (*None or torch.Tensor*) – Consistency weight for each source. Shape needs to be broadcastable to *istft_dec(mag_specgrams)*. We make sure that the weights sum up to 1 along dim *dim*. If *src_weights* is None, compute them based on relative power.
- **dim** (*int*) – Axis which contains the sources in *mag_specgrams*. Used for consistency constraint.

Returns *torch.Tensor* – estimated waveforms of shape (batch, n_src, time).

Examples

```
>>> stft = Encoder(STFTFB(n_filters=256, kernel_size=256, stride=128))
>>> wav = torch.randn(2, 3, 8000)
>>> specs = stft(wav)
>>> masked_specs = specs * torch.sigmoid(torch.randn_like(specs))
>>> mag = transforms.mag(masked_specs, -2)
>>> est_wav = misi(wav.sum(1), mag, stft, n_iter=32)
```

References [1] Gunawan and Sen, "Iterative Phase Estimation for the Synthesis of Separated Sources From Single-Channel Mixtures," in IEEE Signal Processing Letters, 2010.

[2] Wang, LeRoux et al. "End-to-End Speech Separation with Unfolded Iterative Phase Reconstruction." Interspeech 2018 (2018)

10.4.2 Complex transforms

asteroid_filterbanks.transforms.**mul_c**(*inp*, *other*, *dim: int = -2*)

Entrywise product for complex valued tensors.

Operands are assumed to have the real parts of each entry followed by the imaginary parts of each entry along dimension *dim*, e.g. for, `dim = 1`, the matrix

is interpreted as

where *j* is such that *j * j = -1*.

Parameters

- **inp** (*torch.Tensor*) – The first operand with real and imaginary parts concatenated on the *dim* axis.
- **other** (*torch.Tensor*) – The second operand.
- **dim** (*int, optional*) – frequency (or equivalent) dimension along which real and imaginary values are concatenated.

Returns

torch.Tensor – The complex multiplication between *inp* and *other*

For now, it assumes that *other* has the same shape as *inp* along *dim*.

asteroid_filterbanks.transforms.**reim**(*x*, *dim: int = -2*) → Tuple[<sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03caf110>, <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03caf150>]

Returns a tuple (re, im).

Parameters

- **x** (torch.Tensor) – Complex valued tensor.
- **dim** (*int*) – frequency (or equivalent) dimension along which real and imaginary values are concatenated.

asteroid_filterbanks.transforms.**mag**(*x*, *dim: int = -2*, *EPS: float = 1e-08*)

Takes the magnitude of a complex tensor.

The operands is assumed to have the real parts of each entry followed by the imaginary parts of each entry along dimension *dim*, e.g. for, `dim = 1`, the matrix

is interpreted as

where *j* is such that *j * j = -1*.

Parameters

- **x** (torch.Tensor) – Complex valued tensor.
- **dim** (*int*) – frequency (or equivalent) dimension along which real and imaginary values are concatenated.

Returns torch.Tensor – The magnitude of x.

asteroid_filterbanks.transforms.**magreim**(*x*, *dim: int = -2*)

Returns a concatenation of (mag, re, im).

Parameters

- **x** (torch.Tensor) – Complex valued tensor.
- **dim** (*int*) – frequency (or equivalent) dimension along which real and imaginary values are concatenated.

asteroid_filterbanks.transforms.**apply_real_mask**(*tf_rep*, *mask*, *dim: int = -2*)

Applies a real-valued mask to a real-valued representation.

It corresponds to ReIm mask in [1].

Parameters

- **tf_rep** (torch.Tensor) – The time frequency representation to apply the mask to.
- **mask** (torch.Tensor) – The real-valued mask to be applied.
- **dim** (*int*) – Kept to have the same interface with the other ones.

Returns torch.Tensor – *tf_rep* multiplied by the *mask*.

asteroid_filterbanks.transforms.**apply_mag_mask**(*tf_rep*, *mask*, *dim: int = -2*)

Applies a real-valued mask to a complex-valued representation.

If *tf_rep* has 2N elements along *dim*, *mask* has N elements, *mask* is duplicated along *dim* to apply the same mask to both the Re and Im.

tf_rep is assumed to have the real parts of each entry followed by the imaginary parts of each entry along dimension *dim*, e.g. for, `dim = 1`, the matrix

is interpreted as

where *j* is such that *j * j = -1*.

Parameters

- **tf_rep** (`torch.Tensor`) – The time frequency representation to apply the mask to. Re and Im are concatenated along *dim*.
- **mask** (`torch.Tensor`) – The real-valued mask to be applied.
- **dim** (*int*) – The frequency (or equivalent) dimension of both *tf_rep* and *mask* along which real and imaginary values are concatenated.

Returns `torch.Tensor` – *tf_rep* multiplied by the *mask*.

asteroid_filterbanks.transforms.**apply_complex_mask**(*tf_rep*, *mask*, *dim: int = -2*)

Applies a complex-valued mask to a complex-valued representation.

Operands are assumed to have the real parts of each entry followed by the imaginary parts of each entry along dimension *dim*, e.g. for, `dim = 1`, the matrix

is interpreted as

where *j* is such that *j * j = -1*.

Parameters

- **tf_rep** (`torch.Tensor`) – The time frequency representation to apply the mask to.
- **(class** (*mask*) – *torch.Tensor*): The complex-valued mask to be applied.
- **dim** (*int*) – The frequency (or equivalent) dimension of both *tf_rep* an *mask* along which real and imaginary values are concatenated.

Returns `torch.Tensor` – *tf_rep* multiplied by the *mask* in the complex sense.

asteroid_filterbanks.transforms.**is_asteroid_complex**(*tensor*, *dim: int = -2*)

Check if tensor is complex-like in a given dimension.

Parameters

- **tensor** (*torch.Tensor*) – tensor to be checked.
- **dim** (*int*) – the frequency (or equivalent) dimension along which real and imaginary values are concatenated.

Returns True if dimension is even in the specified dimension, otherwise False

asteroid_filterbanks.transforms.**check_complex**(*tensor*, *dim: int = -2*)

Assert that tensor is an Asteroid-style complex in a given dimension.

Parameters

- **tensor** (*torch.Tensor*) – tensor to be checked.
- **dim** (*int*) – the frequency (or equivalent) dimension along which real and imaginary values are concatenated.

Raises AssertionError if dimension is not even in the specified dimension

asteroid_filterbanks.transforms.**to_numpy**(*tensor*, *dim: int = -2*)

Convert complex-like torch tensor to numpy complex array

Parameters

- **tensor** (*torch.Tensor*) – Complex tensor to convert to numpy.
- **dim** (*int, optional*) – the frequency (or equivalent) dimension along which real and imaginary values are concatenated.

Returns numpy.array – Corresponding complex array.

asteroid_filterbanks.transforms.**from_numpy**(*array*, *dim: int = -2*)
Convert complex numpy array to complex-like torch tensor.

Parameters

- **array** (*np.array*) – array to be converted.
- **dim** (*int, optional*) – the frequency (or equivalent) dimension along which real and imaginary values are concatenated.

Returns torch.Tensor – Corresponding torch.Tensor (complex axis in dim ‘dim‘=

asteroid_filterbanks.transforms.**is_torchaudio_complex**(*x*)
Check if tensor is Torchaudio-style complex-like (last dimension is 2).

Parameters **x** (*torch.Tensor*) – tensor to be checked.

Returns True if last dimension is 2, else False.

asteroid_filterbanks.transforms.**check_torchaudio_complex**(*tensor*)
Assert that tensor is Torchaudo-style complex-like (last dimension is 2).

Parameters **tensor** (*torch.Tensor*) – tensor to be checked.

Raises AssertionError if last dimension is != 2.

asteroid_filterbanks.transforms.**to_torchaudio**(*tensor*, *dim: int = -2*)
Converts complex-like torch tensor to torchaudio style complex tensor.

Parameters

- **tensor** (*torch.tensor*) – asteroid-style complex-like torch tensor.
- **dim** (*int, optional*) – the frequency (or equivalent) dimension along which real and imaginary values are concatenated.

Returns torch.Tensor – torchaudio-style complex-like torch tensor.

asteroid_filterbanks.transforms.**from_torchaudio**(*tensor*, *dim: int = -2*)
Converts torchaudio style complex tensor to complex-like torch tensor.

Parameters

- **tensor** (*torch.tensor*) – torchaudio-style complex-like torch tensor.
- **dim** (*int, optional*) – the frequency (or equivalent) dimension along which real and imaginary values are concatenated.

Returns torch.Tensor – asteroid-style complex-like torch tensor.

asteroid_filterbanks.transforms.**to_torch_complex**(*tensor*, *dim: int = -2*)
Converts complex-like torch tensor to native PyTorch complex tensor.

Parameters

- **tensor** (*torch.tensor*) – asteroid-style complex-like torch tensor.
- **dim** (*int, optional*) – the frequency (or equivalent) dimension along which real and imaginary values are concatenated.

Returns torch.Tensor – Pytorch native complex-like torch tensor.

asteroid_filterbanks.transforms.**from_torch_complex**(*tensor*, *dim: int = -2*)
Converts Pytorch native complex tensor to complex-like torch tensor.

Parameters

- **tensor** (*torch.tensor*) – PyTorch native complex-like torch tensor.
- **dim** (*int, optional*) – the frequency (or equivalent) dimension along which real and imaginary values are concatenated.

Returns torch.Tensor – asteroid-style complex-like torch tensor.

asteroid_filterbanks.transforms.**angle**(*tensor*, *dim: int = -2*)
Return the angle of the complex-like torch tensor.

Parameters

- **tensor** (*torch.Tensor*) – the complex tensor from which to extract the phase.
- **dim** (*int, optional*) – the frequency (or equivalent) dimension along which real and imaginary values are concatenated.

Returns torch.Tensor – The counterclockwise angle from the positive real axis on the complex plane in radians.

asteroid_filterbanks.transforms.**from_magphase**(*mag_spec*, *phase*, *dim: int = -2*)
Return a complex-like torch tensor from magnitude and phase components.

Parameters

- **mag_spec** (*torch.tensor*) – magnitude of the tensor.
- **phase** (*torch.tensor*) – angle of the tensor
- **dim** (*int, optional*) – the frequency (or equivalent) dimension along which real and imaginary values are concatenated.

Returns torch.Tensor – The corresponding complex-like torch tensor.

asteroid_filterbanks.transforms.**magphase**(*spec: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03caf650>*, *dim: int = -2*) → Tuple[<sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03caf690>, <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03caf6d0>]
Splits Asteroid complex-like tensor into magnitude and phase.

asteroid_filterbanks.transforms.**centerfreq_correction**(*spec: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03caf710>*, *kernel_size: int*, *stride: int = None*, *dim: int = -2*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03caf790>
Corrects phase from the input spectrogram so that a sinusoid in the middle of a bin keeps the same phase from one frame to the next.

Parameters

- **spec** – Spectrogram tensor of shape (batch, n_freq + 2, frames).
- **kernel_size** (*int*) – Kernel size of the STFT.
- **stride** (*int*) – Stride of the STFT.

- **dim** (*int*) – Only works of dim=-2.

Returns *Tensor* – the input spec with corrected phase.

asteroid_filterbanks.transforms.**phase_centerfreq_correction**(*phase: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03caf7d0>, kernel_size: int, stride: int = None*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03caf810>

Corrects phase so that a sinusoid in the middle of a bin keeps the same phase from one frame to the next.

Parameters

- **phase** – tensor of shape (batch, n_freq//2 + 1, frames)
- **kernel_size** (*int*) – Kernel size of the STFT.
- **stride** (*int*) – Stride of the STFT.

Returns *Tensor* – corrected phase.

CHAPTER 11

DNN building blocks

11.1 Convolutional blocks

class asteroid.masknn.convolutional.**Conv1DBlock**(*in_chan*, *hid_chan*, *skip_out_chan*, *kernel_size*, *padding*, *dilation*, *norm_type='gLN'*, *causal=False*)

Bases: sphinx.ext.autodoc.importer._MockObject

One dimensional convolutional block, as proposed in [1].

Parameters

- **in_chan** (*int*) – Number of input channels.
- **hid_chan** (*int*) – Number of hidden channels in the depth-wise convolution.
- **skip_out_chan** (*int*) – Number of channels in the skip convolution. If 0 or None, *Conv1DBlock* won't have any skip connections. Corresponds to the the block in v1 or the paper. The *forward* return res instead of [res, skip] in this case.
- **kernel_size** (*int*) – Size of the depth-wise convolutional kernel.
- **padding** (*int*) – Padding of the depth-wise convolution.
- **dilation** (*int*) – Dilation of the depth-wise convolution.
- **norm_type** (*str, optional*) – Type of normalization to use. To choose from
  - `'gLN'`: global Layernorm.
  - `'cLN'`: channelwise Layernorm.
  - `'cgLN'`: cumulative global Layernorm.
  - Any norm supported by *get()*
- **causal** (*bool, optional*) – Whether or not the convolutions are causal

References [1] : "Conv-TasNet: Surpassing ideal time-frequency magnitude masking for speech separation" TASLP 2019 Yi Luo, Nima Mesgarani https://arxiv.org/abs/1809.07454

forward(*x*)
Input shape $(batch, feats, seq)$.

class asteroid.masknn.convolutional.**TDConvNet**(*in_chan*, *n_src*, *out_chan=None*, *n_blocks=8*, *n_repeats=3*, *bn_chan=128*, *hid_chan=512*, *skip_chan=128*, *conv_kernel_size=3*, *norm_type='gLN'*, *mask_act='relu'*, *causal=False*)
Bases: sphinx.ext.autodoc.importer._MockObject

Temporal Convolutional network used in ConvTasnet.

Parameters

- **in_chan** (*int*) – Number of input filters.
- **n_src** (*int*) – Number of masks to estimate.
- **out_chan** (*int, optional*) – Number of bins in the estimated masks. If None, *out_chan = in_chan*.
- **n_blocks** (*int, optional*) – Number of convolutional blocks in each repeat. Defaults to 8.
- **n_repeats** (*int, optional*) – Number of repeats. Defaults to 3.
- **bn_chan** (*int, optional*) – Number of channels after the bottleneck.
- **hid_chan** (*int, optional*) – Number of channels in the convolutional blocks.
- **skip_chan** (*int, optional*) – Number of channels in the skip connections. If 0 or None, TDConvNet won't have any skip connections and the masks will be computed from the residual output. Corresponds to the ConvTasnet architecture in v1 or the paper.
- **conv_kernel_size** (*int, optional*) – Kernel size in convolutional blocks.
- **norm_type** (*str, optional*) – To choose from 'BN', 'gLN', 'cLN'.
- **mask_act** (*str, optional*) – Which non-linear function to generate mask.
- **causal** (*bool, optional*) – Whether or not the convolutions are causal.

References [1] : "Conv-TasNet: Surpassing ideal time-frequency magnitude masking for speech separation" TASLP 2019 Yi Luo, Nima Mesgarani https://arxiv.org/abs/1809.07454

forward(*mixture_w*)
Forward.

Parameters **mixture_w** (torch.Tensor) – Tensor of shape $(batch, nfilters, nframes)$

Returns torch.Tensor – estimated mask of shape $(batch, nsrc, nfilters, nframes)$

class asteroid.masknn.convolutional.**TDConvNetpp**(*in_chan*, *n_src*, *out_chan=None*, *n_blocks=8*, *n_repeats=3*, *bn_chan=128*, *hid_chan=512*, *skip_chan=128*, *conv_kernel_size=3*, *norm_type='fgLN'*, *mask_act='relu'*)
Bases: sphinx.ext.autodoc.importer._MockObject

Improved Temporal Convolutional network used in [1] (TDCN++)

Parameters

- **in_chan** (*int*) – Number of input filters.
- **n_src** (*int*) – Number of masks to estimate.
- **out_chan** (*int, optional*) – Number of bins in the estimated masks. If `None`, *out_chan* = *in_chan*.
- **n_blocks** (*int, optional*) – Number of convolutional blocks in each repeat. Defaults to 8.
- **n_repeats** (*int, optional*) – Number of repeats. Defaults to 3.
- **bn_chan** (*int, optional*) – Number of channels after the bottleneck.
- **hid_chan** (*int, optional*) – Number of channels in the convolutional blocks.
- **skip_chan** (*int, optional*) – Number of channels in the skip connections. If 0 or None, TDConvNet won't have any skip connections and the masks will be computed from the residual output. Corresponds to the ConvTasnet architecture in v1 or the paper.
- **kernel_size** (*int, optional*) – Kernel size in convolutional blocks.
- **norm_type** (*str, optional*) – To choose from `'BN'`, `'gLN'`, `'cLN'`.
- **mask_act** (*str, optional*) – Which non-linear function to generate mask.

References [1] : Kavalerov, Ilya et al. "Universal Sound Separation." in WASPAA 2019

Note: The differences wrt to ConvTasnet's TCN are:

1. Channel wise layer norm instead of global
2. Longer-range skip-residual connections from earlier repeat inputs to later repeat inputs after passing them through dense layer.
3. Learnable scaling parameter after each dense layer. The scaling parameter for the second dense layer in each convolutional block (which is applied rightbefore the residual connection) is initialized to an exponentially decaying scalar equal to 0.9**L, where L is the layer or block index.

---

forward(*mixture_w*)
Forward.

Parameters **mixture_w** (`torch.Tensor`) – Tensor of shape $(batch, nfilters, nframes)$

Returns `torch.Tensor` – estimated mask of shape $(batch, nsrc, nfilters, nframes)$

class asteroid.masknn.convolutional.**DCUNetComplexEncoderBlock**(*in_chan*, *out_chan*, *kernel_size*, *stride*, *padding*, *norm_type='bN'*, *activation='leaky_relu'*)

Bases: `sphinx.ext.autodoc.importer._MockObject`

Encoder block as proposed in [1].

Parameters

- **in_chan** (*int*) – Number of input channels.

- **out_chan** (*int*) – Number of output channels.
- **kernel_size** (*Tuple[int, int]*) – Convolution kernel size.
- **stride** (*Tuple[int, int]*) – Convolution stride.
- **padding** (*Tuple[int, int]*) – Convolution padding.
- **norm_type** (*str, optional*) – Type of normalization to use. See *norms* for valid values.
- **activation** (*str, optional*) – Type of activation to use. See `activations` for valid values.

References [1] : "Phase-aware Speech Enhancement with Deep Complex U-Net", Hyeong-Seok Choi et al. https://arxiv.org/abs/1903.03107

class asteroid.masknn.convolutional.**DCUNetComplexDecoderBlock**(*in_chan*, *out_chan*, *kernel_size*, *stride*, *padding*, *output_padding=(0, 0)*, *norm_type='bN'*, *activation='leaky_relu'*)

Bases: `sphinx.ext.autodoc.importer._MockObject`

Decoder block as proposed in [1].

Parameters

- **in_chan** (*int*) – Number of input channels.
- **out_chan** (*int*) – Number of output channels.
- **kernel_size** (*Tuple[int, int]*) – Convolution kernel size.
- **stride** (*Tuple[int, int]*) – Convolution stride.
- **padding** (*Tuple[int, int]*) – Convolution padding.
- **norm_type** (*str, optional*) – Type of normalization to use. See *norms* for valid values.
- **activation** (*str, optional*) – Type of activation to use. See `activations` for valid values.

References [1] : "Phase-aware Speech Enhancement with Deep Complex U-Net", Hyeong-Seok Choi et al. https://arxiv.org/abs/1903.03107

class asteroid.masknn.convolutional.**DCUMaskNet**(*encoders*, *decoders*, *fix_length_mode=None*, ***kwargs*)

Bases: `asteroid.masknn.base.BaseDCUMaskNet`

Masking part of DCUNet, as proposed in [1].

Valid *architecture* values for the `default_architecture` classmethod are: "Large-DCUNet-20", "DCUNet-20", "DCUNet-16", "DCUNet-10" and "mini".

Valid *fix_length_mode* values are [None, "pad", "trim"].

Input shape is expected to be $(batch, nfreqs, time)$, with $nfreqs - 1$ divisible by $f_0 * f_1 * ... * f_N$ where f_k are the frequency strides of the encoders, and $time - 1$ is divisible by $t_0 * t_1 * ... * t_N$ where t_N are the time strides of the encoders.

References [1] : "Phase-aware Speech Enhancement with Deep Complex U-Net", Hyeong-Seok Choi et al. https://arxiv.org/abs/1903.03107

fix_input_dims(*x*)
Overwrite this in subclasses to implement input dimension checks.

fix_output_dims(*out*, *x*)
Overwrite this in subclasses to implement output dimension checks. y is the output and x was the input (passed to use the shape).

class asteroid.masknn.convolutional.**SuDORMRF**(*in_chan*, *n_src*, *bn_chan=128*, *num_blocks=16*, *upsampling_depth=4*, *mask_act='softmax'*)

Bases: sphinx.ext.autodoc.importer._MockObject

SuDORMRF mask network, as described in [1].

Parameters

- **in_chan** (*int*) – Number of input channels. Also number of output channels.
- **n_src** (*int*) – Number of sources in the input mixtures.
- **bn_chan** (*int, optional*) – Number of bins in the bottleneck layer and the UNet blocks.
- **num_blocks** (*int*) – Number of of UBlocks.
- **upsampling_depth** (*int*) – Depth of upsampling.
- **mask_act** (*str*) – Name of output activation.

References [1] : "Sudo rm -rf: Efficient Networks for Universal Audio Source Separation", Tzinis et al. MLSP 2020.

class asteroid.masknn.convolutional.**SuDORMRFImproved**(*in_chan*, *n_src*, *bn_chan=128*, *num_blocks=16*, *upsampling_depth=4*, *mask_act='relu'*)

Bases: sphinx.ext.autodoc.importer._MockObject

Improved SuDORMRF mask network, as described in [1].

Parameters

- **in_chan** (*int*) – Number of input channels. Also number of output channels.
- **n_src** (*int*) – Number of sources in the input mixtures.
- **bn_chan** (*int, optional*) – Number of bins in the bottleneck layer and the UNet blocks.
- **num_blocks** (*int*) – Number of of UBlocks
- **upsampling_depth** (*int*) – Depth of upsampling
- **mask_act** (*str*) – Name of output activation.

References [1] : "Sudo rm -rf: Efficient Networks for Universal Audio Source Separation", Tzinis et al. MLSP 2020.

class asteroid.masknn.convolutional.**UBlock**(*out_chan=128*, *in_chan=512*, *upsampling_depth=4*)

Bases: asteroid.masknn.convolutional._BaseUBlock

Upsampling block.

Based on the following principle: REDUCE ---> SPLIT ---> TRANSFORM --> MERGE

forward(*x*)

Parameters **x** – input feature map

Returns transformed feature map

class asteroid.masknn.convolutional.**UConvBlock**(*out_chan=128*, *in_chan=512*, *upsampling_depth=4*)

Bases: asteroid.masknn.convolutional._BaseUBlock

Block which performs successive downsampling and upsampling in order to be able to analyze the input features in multiple resolutions.

forward(*x*)

Args x: input feature map

Returns transformed feature map

11.2 Recurrent blocks

class asteroid.masknn.recurrent.**SingleRNN**(*rnn_type*, *input_size*, *hidden_size*, *n_layers=1*, *dropout=0*, *bidirectional=False*)

Bases: sphinx.ext.autodoc.importer._MockObject

Module for a RNN block.

Inspired from https://github.com/yluo42/TAC/blob/master/utility/models.py Licensed under CC BY-NC-SA 3.0 US.

Parameters

- **rnn_type** (*str*) – Select from 'RNN', 'LSTM', 'GRU'. Can also be passed in lowercase letters.
- **input_size** (*int*) – Dimension of the input feature. The input should have shape [batch, seq_len, input_size].
- **hidden_size** (*int*) – Dimension of the hidden state.
- **n_layers** (*int, optional*) – Number of layers used in RNN. Default is 1.
- **dropout** (*float, optional*) – Dropout ratio. Default is 0.
- **bidirectional** (*bool, optional*) – Whether the RNN layers are bidirectional. Default is False.

forward(*inp*)

Input shape [batch, seq, feats]

class asteroid.masknn.recurrent.**MulCatRNN**(*rnn_type*, *input_size*, *hidden_size*, *n_layers=1*, *dropout=0*, *bidirectional=False*)

Bases: sphinx.ext.autodoc.importer._MockObject

MulCat RNN block from [1].

Composed of two RNNs, returns `cat([RNN_1(x) * RNN_2(x), x])`.

Parameters

- **rnn_type** (*str*) – Select from `'RNN'`, `'LSTM'`, `'GRU'`. Can also be passed in lowercase letters.
- **input_size** (*int*) – Dimension of the input feature. The input should have shape [batch, seq_len, input_size].
- **hidden_size** (*int*) – Dimension of the hidden state.
- **n_layers** (*int, optional*) – Number of layers used in RNN. Default is 1.
- **dropout** (*float, optional*) – Dropout ratio. Default is 0.
- **bidirectional** (*bool, optional*) – Whether the RNN layers are bidirectional. Default is `False`.

References [1] Eliya Nachmani, Yossi Adi, & Lior Wolf. (2020). Voice Separation with an Unknown Number of Multiple Speakers.

forward(*inp*)

Input shape [batch, seq, feats]

class asteroid.masknn.recurrent.**StackedResidualRNN**(*rnn_type*, *n_units*, *n_layers=4*, *dropout=0.0*, *bidirectional=False*)

Bases: `sphinx.ext.autodoc.importer._MockObject`

Stacked RNN with builtin residual connection. Only supports forward RNNs. See StackedResidualBiRNN for bidirectional ones.

Parameters

- **rnn_type** (*str*) – Select from `'RNN'`, `'LSTM'`, `'GRU'`. Can also be passed in lowercase letters.
- **n_units** (*int*) – Number of units in recurrent layers. This will also be the expected input size.
- **n_layers** (*int*) – Number of recurrent layers.
- **dropout** (*float*) – Dropout value, between 0. and 1. (Default: 0.)
- **bidirectional** (*bool*) – If True, use bidirectional RNN, else unidirectional. (Default: False)

forward(*x*)

Builtin residual connections + dropout applied before residual. Input shape : [batch, time_axis, feat_axis]

class asteroid.masknn.recurrent.**StackedResidualBiRNN**(*rnn_type*, *n_units*, *n_layers=4*, *dropout=0.0*, *bidirectional=True*)

Bases: `sphinx.ext.autodoc.importer._MockObject`

Stacked Bidirectional RNN with builtin residual connection. Residual connections are applied on both RNN directions. Only supports bidiriectional RNNs. See StackedResidualRNN for unidirectional ones.

Parameters

- **rnn_type** (*str*) – Select from `'RNN'`, `'LSTM'`, `'GRU'`. Can also be passed in lowercase letters.

- **n_units** (*int*) – Number of units in recurrent layers. This will also be the expected input size.
- **n_layers** (*int*) – Number of recurrent layers.
- **dropout** (*float*) – Dropout value, between 0. and 1. (Default: 0.)
- **bidirectional** (*bool*) – If True, use bidirectional RNN, else unidirectional. (Default: False)

forward(*x*)

Builtin residual connections + dropout applied before residual. Input shape : [batch, time_axis, feat_axis]

class asteroid.masknn.recurrent.**DPRNNBlock**(*in_chan*, *hid_size*, *norm_type='gLN'*, *bidirectional=True*, *rnn_type='LSTM'*, *use_mulcat=False*, *num_layers=1*, *dropout=0*)

Bases: sphinx.ext.autodoc.importer._MockObject

Dual-Path RNN Block as proposed in [1].

Parameters

- **in_chan** (*int*) – Number of input channels.
- **hid_size** (*int*) – Number of hidden neurons in the RNNs.
- **norm_type** (*str, optional*) – Type of normalization to use. To choose from - `'gLN'`: global Layernorm - `'cLN'`: channelwise Layernorm
- **bidirectional** (*bool, optional*) – True for bidirectional Inter-Chunk RNN.
- **rnn_type** (*str, optional*) – Type of RNN used. Choose from `'RNN'`, `'LSTM'` and `'GRU'`.
- **num_layers** (*int, optional*) – Number of layers used in each RNN.
- **dropout** (*float, optional*) – Dropout ratio. Must be in [0, 1].

References [1] "Dual-path RNN: efficient long sequence modeling for time-domain single-channel speech separation", Yi Luo, Zhuo Chen and Takuya Yoshioka. https://arxiv.org/abs/1910.06379

forward(*x*)

Input shape : [batch, feats, chunk_size, num_chunks]

class asteroid.masknn.recurrent.**DPRNN**(*in_chan*, *n_src*, *out_chan=None*, *bn_chan=128*, *hid_size=128*, *chunk_size=100*, *hop_size=None*, *n_repeats=6*, *norm_type='gLN'*, *mask_act='relu'*, *bidirectional=True*, *rnn_type='LSTM'*, *use_mulcat=False*, *num_layers=1*, *dropout=0*)

Bases: sphinx.ext.autodoc.importer._MockObject

Dual-path RNN Network for Single-Channel Source Separation introduced in [1].

Parameters

- **in_chan** (*int*) – Number of input filters.
- **n_src** (*int*) – Number of masks to estimate.
- **out_chan** (*int or None*) – Number of bins in the estimated masks. Defaults to *in_chan*.
- **bn_chan** (*int*) – Number of channels after the bottleneck. Defaults to 128.

- **hid_size** (*int*) – Number of neurons in the RNNs cell state. Defaults to 128.
- **chunk_size** (*int*) – window size of overlap and add processing. Defaults to 100.
- **hop_size** (*int or None*) – hop size (stride) of overlap and add processing. Default to *chunk_size // 2* (50% overlap).
- **n_repeats** (*int*) – Number of repeats. Defaults to 6.
- **norm_type** (*str, optional*) – Type of normalization to use. To choose from
  - `'gLN'`: global Layernorm
  - `'cLN'`: channelwise Layernorm
- **mask_act** (*str, optional*) – Which non-linear function to generate mask.
- **bidirectional** (*bool, optional*) – True for bidirectional Inter-Chunk RNN (Intra-Chunk is always bidirectional).
- **rnn_type** (*str, optional*) – Type of RNN used. Choose between `'RNN'`, `'LSTM'` and `'GRU'`.
- **num_layers** (*int, optional*) – Number of layers in each RNN.
- **dropout** (*float, optional*) – Dropout ratio, must be in [0,1].

References [1] "Dual-path RNN: efficient long sequence modeling for time-domain single-channel speech separation", Yi Luo, Zhuo Chen and Takuya Yoshioka. https://arxiv.org/abs/1910.06379

forward(*mixture_w*)

Forward.

Parameters **mixture_w** (`torch.Tensor`) – Tensor of shape $(batch, nfilters, nframes)$

Returns `torch.Tensor` – estimated mask of shape $(batch, nsrc, nfilters, nframes)$

class asteroid.masknn.recurrent.**LSTMMasker**(*in_chan*, *n_src*, *out_chan=None*, *rnn_type='lstm'*, *n_layers=4*, *hid_size=512*, *dropout=0.3*, *mask_act='sigmoid'*, *bidirectional=True*)

Bases: `sphinx.ext.autodoc.importer._MockObject`

LSTM mask network introduced in [1], without skip connections.

Parameters

- **in_chan** (*int*) – Number of input filters.
- **n_src** (*int*) – Number of masks to estimate.
- **out_chan** (*int or None*) – Number of bins in the estimated masks. Defaults to *in_chan*.
- **rnn_type** (*str, optional*) – Type of RNN used. Choose between `'RNN'`, `'LSTM'` and `'GRU'`.
- **n_layers** (*int, optional*) – Number of layers in each RNN.
- **hid_size** (*int*) – Number of neurons in the RNNs cell state.
- **mask_act** (*str, optional*) – Which non-linear function to generate mask.
- **bidirectional** (*bool, optional*) – Whether to use BiLSTM
- **dropout** (*float, optional*) – Dropout ratio, must be in [0,1].

References [1]: Yi Luo et al. "Real-time Single-channel Dereverberation and Separation with Time-domain Audio Separation Network", Interspeech 2018

class asteroid.masknn.recurrent.**DCCRMaskNetRNN**(*in_size*, *hid_size=128*, *rnn_type='LSTM'*, *n_layers=2*, *norm_type=None*, ***rnn_kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

RNN (LSTM) layer between encoders and decoders introduced in [1].

Parameters

- **in_size** (*int*) – Number of inputs to the RNN. Must be the product of non-batch, non-time dimensions of output shape of last encoder, i.e. if the last encoder output shape is $(batch, nchans, nfreqs, time)$, *in_size* must be $nchans * nfreqs$.
- **hid_size** (*int, optional*) – Number of units in RNN.
- **rnn_type** (*str, optional*) – Type of RNN to use. See SingleRNN for valid values.
- **n_layers** (*int, optional*) – Number of layers used in RNN.
- **norm_type** (*Optional[str], optional*) – Norm to use after linear. See asteroid.masknn.norms for valid values. (Not used in [1]).
- **rnn_kwargs** (*optional*) – Passed to SingleRNN().

References [1] : "DCCRN: Deep Complex Convolution Recurrent Network for Phase-Aware Speech Enhancement", Yanxin Hu et al. https://arxiv.org/abs/2008.00264

forward(*x: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c26850>*)

Input shape: [batch, . . . , time]

class asteroid.masknn.recurrent.**DCCRMaskNet**(*encoders*, *decoders*, *n_freqs*, ***kwargs*)

Bases: asteroid.masknn.base.BaseDCUMaskNet

Masking part of DCCRNet, as proposed in [1].

Valid *architecture* values for the default_architecture classmethod are: "DCCRN" and "mini".

Parameters

- **encoders** (list of length *N* of tuples of (in_chan, out_chan, kernel_size, stride, padding)) – Arguments of encoders of the u-net
- **decoders** (list of length *N* of tuples of (in_chan, out_chan, kernel_size, stride, padding)) – Arguments of decoders of the u-net
- **n_freqs** (*int*) – Number of frequencies (dim 1) of input to .forward(). Must be divisible by $f_0 * f_1 * . . . * f_N$ where f_k are the frequency strides of the encoders.

Input shape is expected to be $(batch, nfreqs, time)$, with $nfreqs$ divisible by $f_0 * f_1 * . . . * f_N$ where f_k are the frequency strides of the encoders.

References [1] : "DCCRN: Deep Complex Convolution Recurrent Network for Phase-Aware Speech Enhancement", Yanxin Hu et al. https://arxiv.org/abs/2008.00264

fix_input_dims(*x*)

Overwrite this in subclasses to implement input dimension checks.

11.3 Attention blocks

class asteroid.masknn.attention.**ImprovedTransformedLayer**(*embed_dim*, *n_heads*, *dim_ff*, *dropout=0.0*, *activation='relu'*, *bidirectional=True*, *norm='gLN'*)

Bases: sphinx.ext.autodoc.importer._MockObject

Improved Transformer module as used in [1]. It is Multi-Head self-attention followed by LSTM, activation and linear projection layer.

Parameters

- **embed_dim** (*int*) – Number of input channels.
- **n_heads** (*int*) – Number of attention heads.
- **dim_ff** (*int*) – Number of neurons in the RNNs cell state. Defaults to 256. RNN here replaces standard FF linear layer in plain Transformer.
- **dropout** (*float, optional*) – Dropout ratio, must be in [0,1].
- **activation** (*str, optional*) – activation function applied at the output of RNN.
- **bidirectional** (*bool, optional*) – True for bidirectional Inter-Chunk RNN (Intra-Chunk is always bidirectional).
- **norm** (*str, optional*) – Type of normalization to use.

References [1] Chen, Jingjing, Qirong Mao, and Dong Liu. "Dual-Path Transformer Network: Direct Context-Aware Modeling for End-to-End Monaural Speech Separation." arXiv (2020).

class asteroid.masknn.attention.**DPTransformer**(*in_chan*, *n_src*, *n_heads=4*, *ff_hid=256*, *chunk_size=100*, *hop_size=None*, *n_repeats=6*, *norm_type='gLN'*, *ff_activation='relu'*, *mask_act='relu'*, *bidirectional=True*, *dropout=0*)

Bases: sphinx.ext.autodoc.importer._MockObject

Dual-path Transformer introduced in [1].

Parameters

- **in_chan** (*int*) – Number of input filters.
- **n_src** (*int*) – Number of masks to estimate.
- **n_heads** (*int*) – Number of attention heads.
- **ff_hid** (*int*) – Number of neurons in the RNNs cell state. Defaults to 256.
- **chunk_size** (*int*) – window size of overlap and add processing. Defaults to 100.
- **hop_size** (*int or None*) – hop size (stride) of overlap and add processing. Default to *chunk_size // 2* (50% overlap).
- **n_repeats** (*int*) – Number of repeats. Defaults to 6.
- **norm_type** (*str, optional*) – Type of normalization to use.
- **ff_activation** (*str, optional*) – activation function applied at the output of RNN.

- **mask_act** (*str, optional*) – Which non-linear function to generate mask.
- **bidirectional** (*bool, optional*) – True for bidirectional Inter-Chunk RNN (Intra-Chunk is always bidirectional).
- **dropout** (*float, optional*) – Dropout ratio, must be in [0,1].

References [1] Chen, Jingjing, Qirong Mao, and Dong Liu. "Dual-Path Transformer Network: Direct Context-Aware Modeling for End-to-End Monaural Speech Separation." arXiv (2020).

forward(*mixture_w*)
Forward.

Parameters **mixture_w** (torch.Tensor) – Tensor of shape $(batch, nfilters, nframes)$

Returns torch.Tensor – estimated mask of shape $(batch, nsrc, nfilters, nframes)$

11.4 Norms

class asteroid.masknn.norms.**GlobLN**(*channel_size*)
Bases: asteroid.masknn.norms._LayerNorm

Global Layer Normalization (globLN).

forward(*x*, *EPS: float = 1e-08*)
Applies forward pass.

Works for any input size > 2D.

Parameters **x** (torch.Tensor) – Shape *[batch, chan, *]*

Returns torch.Tensor – gLN_x *[batch, chan, *]*

class asteroid.masknn.norms.**ChanLN**(*channel_size*)
Bases: asteroid.masknn.norms._LayerNorm

Channel-wise Layer Normalization (chanLN).

forward(*x*, *EPS: float = 1e-08*)
Applies forward pass.

Works for any input size > 2D.

Parameters **x** (torch.Tensor) – *[batch, chan, *]*

Returns torch.Tensor – chanLN_x *[batch, chan, *]*

class asteroid.masknn.norms.**CumLN**(*channel_size*)
Bases: asteroid.masknn.norms._LayerNorm

Cumulative Global layer normalization(cumLN).

forward(*x*, *EPS: float = 1e-08*)

Parameters **x** (torch.Tensor) – Shape *[batch, channels, length]*

Returns torch.Tensor – cumLN_x *[batch, channels, length]*

class asteroid.masknn.norms.**FeatsGlobLN**(*channel_size*)
Bases: asteroid.masknn.norms._LayerNorm

Feature-wise global Layer Normalization (FeatsGlobLN). Applies normalization over frames for each channel.

forward(*x*, *EPS: float = 1e-08*)
Applies forward pass.

Works for any input size > 2D.

Parameters **x** (torch.Tensor) – *[batch, chan, time]*

Returns torch.Tensor – chanLN_x *[batch, chan, time]*

class asteroid.masknn.norms.**BatchNorm**(**args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Wrapper class for pytorch BatchNorm1D and BatchNorm2D

asteroid.masknn.norms.**gLN**
alias of *asteroid.masknn.norms.GlobLN*

asteroid.masknn.norms.**fgLN**
alias of *asteroid.masknn.norms.FeatsGlobLN*

asteroid.masknn.norms.**cLN**
alias of *asteroid.masknn.norms.ChanLN*

asteroid.masknn.norms.**cgLN**
alias of *asteroid.masknn.norms.CumLN*

asteroid.masknn.norms.**bN**
alias of *asteroid.masknn.norms.BatchNorm*

asteroid.masknn.norms.**register_norm**(*custom_norm*)
Register a custom norm, gettable with *norms.get*.

Parameters **custom_norm** – Custom norm to register.

asteroid.masknn.norms.**get**(*identifier*)
Returns a norm class from a string. Returns its input if it is callable (already a _LayerNorm for example).

Parameters **identifier** (*str or Callable or None*) – the norm identifier.

Returns _LayerNorm or None

asteroid.masknn.norms.**get_complex**(*identifier*)
Like *.get* but returns a complex norm created with *asteroid.complex_nn.OnReIm*.

11.5 Complex number support

Complex building blocks that work with PyTorch native (!) complex tensors, i.e. dtypes complex64/complex128, or tensors for which *.is_complex()* returns True.

Note that Asteroid code has two other representations of complex numbers:

- Torchaudio representation [. . . , 2] where [. . . , 0] and [. . . , 1] are real and imaginary components, respectively
- Asteroid style representation, identical to the Torchaudio representation, but with the last dimension concatenated: tensor([r1, r2, . . . , rn, i1, i2, . . . , in]). The concatenated (2 * n) dimension may be at an arbitrary position, i.e. the tensor is of shape [. . . , 2 * n, . . .]. See *asteroid_filterbanks.transforms* for details.

asteroid.complex_nn.**on_reim**(*f*)
Make a complex-valued function callable from a real-valued one by applying it to the real and imaginary components independently.

Returns cf(x), complex version of *f* – A function that applies *f* to the real and imaginary components of *x* and returns the result as PyTorch complex tensor.

class asteroid.complex_nn.**OnReIm**(*module_cls*, **args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Like *on_reim*, but for stateful modules.

Parameters **module_cls** (*callable*) – A class or function that returns a Torch module/functional. Called 2x with *args, **kwargs, to construct the real and imaginary component modules.

class asteroid.complex_nn.**ComplexMultiplicationWrapper**(*module_cls*, **args*, ***kwargs*)
Bases: sphinx.ext.autodoc.importer._MockObject

Make a complex-valued module *F* from a real-valued module *f* by applying complex multiplication rules:

F(a + i b) = f1(a) - f1(b) + i (f2(b) + f2(a))

where *f1*, *f2* are instances of *f* that do *not* share weights.

Parameters **module_cls** (*callable*) – A class or function that returns a Torch module/functional. Constructor of *f* in the formula above. Called 2x with **args*, ***kwargs*, to construct the real and imaginary component modules.

class asteroid.complex_nn.**ComplexSingleRNN**(*rnn_type*, *input_size*, *hidden_size*, *n_layers=1*, *dropout=0*, *bidirectional=False*)
Bases: sphinx.ext.autodoc.importer._MockObject

Module for a complex RNN block.

This is similar to **:cls:‘asteroid.masknn.recurrent.SingleRNN‘** but uses complex multiplication as described in [1]. Arguments are identical to those of *SingleRNN*, except for *dropout*, which is not yet supported.

Parameters

- **rnn_type** (*str*) – Select from `'RNN'`, `'LSTM'`, `'GRU'`. Can also be passed in lowercase letters.
- **input_size** (*int*) – Dimension of the input feature. The input should have shape [batch, seq_len, input_size].
- **hidden_size** (*int*) – Dimension of the hidden state.
- **n_layers** (*int, optional*) – Number of layers used in RNN. Default is 1.
- **bidirectional** (*bool, optional*) – Whether the RNN layers are bidirectional. Default is `False`.
- **dropout** – Not yet supported.

References [1] : “DCCRN: Deep Complex Convolution Recurrent Network for Phase-Aware Speech Enhancement”, Yanxin Hu et al. https://arxiv.org/abs/2008.00264

forward(*x:* <*sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c26850*>) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c26850>
Input shape [batch, seq, feats]

class asteroid.complex_nn.**BoundComplexMask**(*bound_type*)
Bases: sphinx.ext.autodoc.importer._MockObject

Module version of *bound_complex_mask*

asteroid.complex_nn.**bound_complex_mask**(*mask:* <*sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c26850*>, *bound_type='tanh'*)
Bound a complex mask, as proposed in [1], section 3.2.

Valid bound types, for a complex mask $M = |M|e^{i(M)}$:

- Unbounded ("UBD"): $M_{\text{UBD}} = M$
- Sigmoid ("BDSS"): $M_{\text{BDSS}} = (|M|)e^{i((M))}$
- Tanh ("BDT"): $M_{\text{BDT}} = \tanh(|M|)e^{i(M)}$

Parameters **bound_type** (*str or None*) – The type of bound to use, either of "tanh"/"bdt" (default), "sigmoid"/"bdss" or None/"bdt".

References [1] : "Phase-aware Speech Enhancement with Deep Complex U-Net", Hyeong-Seok Choi et al. https://arxiv.org/abs/1903.03107

CHAPTER 12

Models

12.1 Base classes

class asteroid.models.base_models.**BaseModel**(*sample_rate: float*, *in_channels: Optional[int] = 1*)

Bases: sphinx.ext.autodoc.importer._MockObject

Base class for serializable models.

Defines saving/loading procedures, and separation interface to *separate*. Need to overwrite the *forward* and *get_model_args* methods.

Models inheriting from *BaseModel* can be used by asteroid.separate and by the *asteroid-infer* CLI. For models whose *forward* doesn't go from waveform to waveform tensors, overwrite *forward_wav* to return waveform tensors.

Parameters

- **sample_rate** (*float*) – Operating sample rate of the model.
- **in_channels** – Number of input channels in the signal. If None, no checks will be performed.

sample_rate
Operating sample rate of the model (float).

separate(**args*, ***kwargs*)
Convenience for separate().

torch_separate(**args*, ***kwargs*)
Convenience for torch_separate().

numpy_separate(**args*, ***kwargs*)
Convenience for numpy_separate().

file_separate(**args*, ***kwargs*)
Convenience for file_separate().

forward_wav(*wav*, **args*, ***kwargs*)
Separation method for waveforms.

In case the network's *forward* doesn't have waveforms as input/output, overwrite this method to separate from waveform to waveform. Should return a single torch.Tensor, the separated waveforms.

Parameters **wav** (*torch.Tensor*) – waveform array/tensor. Shape: 1D, 2D or 3D tensor, time last.

classmethod from_pretrained(*pretrained_model_conf_or_path*, **args*, ***kwargs*)
Instantiate separation model from a model config (file or dict).

Parameters

- **pretrained_model_conf_or_path** (*Union[dict, str]*) – model conf as returned by *serialize*, or path to it. Need to contain *model_args* and *state_dict* keys.
- ***args** – Positional arguments to be passed to the model.
- ****kwargs** – Keyword arguments to be passed to the model. They overwrite the ones in the model package.

Returns nn.Module corresponding to the pretrained model conf/URL.

Raises *ValueError if the input config file doesn't contain the keys – model_name*, *model_args* or *state_dict*.

serialize()
Serialize model and output dictionary.

Returns dict, serialized model with keys *model_args* and *state_dict*.

get_state_dict()
In case the state dict needs to be modified before sharing the model.

get_model_args()
Should return args to re-instantiate the class.

class asteroid.models.base_models.**BaseEncoderMaskerDecoder**(*encoder*, *masker*, *decoder*, *encoder_activation=None*)
Bases: *asteroid.models.base_models.BaseModel*

Base class for encoder-masker-decoder separation models.

Parameters

- **encoder** (Encoder) – Encoder instance.
- **masker** (*nn.Module*) – masker network.
- **decoder** (Decoder) – Decoder instance.
- **encoder_activation** (*Optional[str], optional*) – Activation to apply after encoder. See `asteroid.masknn.activations` for valid values.

forward(*wav*)
Enc/Mask/Dec model forward

Parameters **wav** (*torch.Tensor*) – waveform tensor. 1D, 2D or 3D tensor, time last.

Returns torch.Tensor, of shape (batch, n_src, time) or (n_src, time).

forward_encoder(*wav: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03bdd1d0>*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03bdd210>
Computes time-frequency representation of *wav*.

Parameters **wav** (*torch.Tensor*) – waveform tensor in 3D shape, time last.

Returns torch.Tensor, of shape (batch, feat, seq).

forward_masker(*tf_rep: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03bdd250>*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03bdd290>
Estimates masks from time-frequency representation.

Parameters **tf_rep** (*torch.Tensor*) – Time-frequency representation in (batch, feat, seq).

Returns *torch.Tensor* – Estimated masks

apply_masks(*tf_rep: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03bdd2d0>*, *est_masks: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03bdd310>*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03bdd350>
Applies masks to time-frequency representation.

Parameters

- **tf_rep** (*torch.Tensor*) – Time-frequency representation in (batch, feat, seq) shape.
- **est_masks** (*torch.Tensor*) – Estimated masks.

Returns *torch.Tensor* – Masked time-frequency representations.

forward_decoder(*masked_tf_rep: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03bdd390>*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03bdd3d0>
Reconstructs time-domain waveforms from masked representations.

Parameters **masked_tf_rep** (*torch.Tensor*) – Masked time-frequency representation.

Returns *torch.Tensor* – Time-domain waveforms.

get_model_args()
Arguments needed to re-instantiate the model.

asteroid.models.base_models.**BaseTasNet**
alias of *asteroid.models.base_models.BaseEncoderMaskerDecoder*

12.2 Ready-to-use models

class asteroid.models.conv_tasnet.**ConvTasNet**(*n_src*, *out_chan=None*, *n_blocks=8*, *n_repeats=3*, *bn_chan=128*, *hid_chan=512*, *skip_chan=128*, *conv_kernel_size=3*, *norm_type='gLN'*, *mask_act='sigmoid'*, *in_chan=None*, *causal=False*, *fb_name='free'*, *kernel_size=16*, *n_filters=512*, *stride=8*, *encoder_activation=None*, *sample_rate=8000*, ***fb_kwargs*)
Bases: *asteroid.models.base_models.BaseEncoderMaskerDecoder*

ConvTasNet separation model, as described in [1].

Parameters

- **n_src** (*int*) – Number of sources in the input mixtures.
- **out_chan** (*int, optional*) – Number of bins in the estimated masks. If `None`, *out_chan = in_chan*.

- **n_blocks** (*int, optional*) – Number of convolutional blocks in each repeat. Defaults to 8.
- **n_repeats** (*int, optional*) – Number of repeats. Defaults to 3.
- **bn_chan** (*int, optional*) – Number of channels after the bottleneck.
- **hid_chan** (*int, optional*) – Number of channels in the convolutional blocks.
- **skip_chan** (*int, optional*) – Number of channels in the skip connections. If 0 or None, TDConvNet won't have any skip connections and the masks will be computed from the residual output. Corresponds to the ConvTasnet architecture in v1 or the paper.
- **conv_kernel_size** (*int, optional*) – Kernel size in convolutional blocks.
- **norm_type** (*str, optional*) – To choose from `'BN'`, `'gLN'`, `'cLN'`.
- **mask_act** (*str, optional*) – Which non-linear function to generate mask.
- **in_chan** (*int, optional*) – Number of input channels, should be equal to n_filters.
- **causal** (*bool, optional*) – Whether or not the convolutions are causal.
- **fb_name** (*str, className*) – Filterbank family from which to make encoder and decoder. To choose among [`'free'`, `'analytic_free'`, `'param_sinc'`, `'stft'`].
- **n_filters** (*int*) – Number of filters / Input dimension of the masker net.
- **kernel_size** (*int*) – Length of the filters.
- **stride** (*int, optional*) – Stride of the convolution. If None (default), set to `kernel_size // 2`.
- **sample_rate** (*float*) – Sampling rate of the model.
- ****fb_kwargs** (*dict*) – Additional kwards to pass to the filterbank creation.

References

- [1] : "Conv-TasNet: Surpassing ideal time-frequency magnitude masking for speech separation" TASLP 2019 Yi Luo, Nima Mesgarani https://arxiv.org/abs/1809.07454

class asteroid.models.conv_tasnet.**VADNet**(*n_src*, *out_chan=None*, *n_blocks=8*, *n_repeats=3*, *bn_chan=128*, *hid_chan=512*, *skip_chan=128*, *conv_kernel_size=3*, *norm_type='gLN'*, *mask_act='sigmoid'*, *in_chan=None*, *causal=False*, *fb_name='free'*, *kernel_size=16*, *n_filters=512*, *stride=8*, *encoder_activation=None*, *sample_rate=8000*, ***fb_kwargs*)

Bases: *asteroid.models.conv_tasnet.ConvTasNet*

forward_decoder(*masked_tf_rep: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03bdd750>*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03bdd790>

Reconstructs time-domain waveforms from masked representations.

Parameters **masked_tf_rep** (*torch.Tensor*) – Masked time-frequency representation.

Returns *torch.Tensor* – Time-domain waveforms.

class asteroid.models.dccrnet.**DCCRNet**(**args*, *stft_n_filters=512*, *stft_kernel_size=400*, *stft_stride=100*, ***masknet_kwargs*)

Bases: *asteroid.models.dcunet.BaseDCUNet*

DCCRNet as proposed in [1].

Parameters

- **architecture** (*str*) – The architecture to use, must be "DCCRN-CL".
- **stft_kernel_size** (*int*) – STFT frame length to use
- **stft_stride** (*int, optional*) – STFT hop length to use.
- **sample_rate** (*float*) – Sampling rate of the model.
- **masknet_kwargs** (*optional*) – Passed to `DCCRMaskNet`

References

- [1] : "DCCRN: Deep Complex Convolution Recurrent Network for Phase-Aware Speech Enhancement", Yanxin Hu et al. https://arxiv.org/abs/2008.00264

masknet_class
alias of *asteroid.masknn.recurrent.DCCRMaskNet*

forward_encoder(*wav*)
Computes time-frequency representation of *wav*.

Parameters **wav** (*torch.Tensor*) – waveform tensor in 3D shape, time last.

Returns torch.Tensor, of shape (batch, feat, seq).

apply_masks(*tf_rep*, *est_masks*)
Applies masks to time-frequency representation.

Parameters

- **tf_rep** (*torch.Tensor*) – Time-frequency representation in (batch, feat, seq) shape.
- **est_masks** (*torch.Tensor*) – Estimated masks.

Returns *torch.Tensor* – Masked time-frequency representations.

class asteroid.models.dcunet.**BaseDCUNet**(*architecture*, *stft_n_filters=1024*, *stft_kernel_size=1024*, *stft_stride=256*, *sample_rate=16000.0*, ***masknet_kwargs*)
Bases: *asteroid.models.base_models.BaseEncoderMaskerDecoder*

Base class for `DCUNet` and `DCCRNet` classes.

Parameters

- **architecture** (*str*) – The architecture to use. Overriden by subclasses.
- **stft_n_filters** (*int*) –
- **stft_kernel_size** (*int*) – STFT frame length to use.
- **stft_stride** (*int, optional*) – STFT hop length to use.
- **sample_rate** (*float*) – Sampling rate of the model.
- **masknet_kwargs** (*optional*) – Passed to the masknet constructor.

forward_encoder(*wav*)
Computes time-frequency representation of *wav*.

Parameters **wav** (*torch.Tensor*) – waveform tensor in 3D shape, time last.

Returns torch.Tensor, of shape (batch, feat, seq).

apply_masks(*tf_rep*, *est_masks*)
Applies masks to time-frequency representation.

Parameters

- **tf_rep** (*torch.Tensor*) – Time-frequency representation in (batch, feat, seq) shape.
- **est_masks** (*torch.Tensor*) – Estimated masks.

Returns *torch.Tensor* – Masked time-frequency representations.

get_model_args()
Arguments needed to re-instantiate the model.

class asteroid.models.dcunet.**DCUNet**(*architecture*, *stft_n_filters=1024*, *stft_kernel_size=1024*, *stft_stride=256*, *sample_rate=16000.0*, ***masknet_kwargs*)
Bases: *asteroid.models.dcunet.BaseDCUNet*

DCUNet as proposed in [1].

Parameters

- **architecture** (*str*) – The architecture to use, any of "DCUNet-10", "DCUNet-16", "DCUNet-20", "Large-DCUNet-20".
- **stft_n_filters** (*int*) –
- **stft_kernel_size** (*int*) – STFT frame length to use.
- **stft_stride** (*int, optional*) – STFT hop length to use.
- **sample_rate** (*float*) – Sampling rate of the model.
- **masknet_kwargs** (*optional*) – Passed to DCUMaskNet

References

- [1] : "Phase-aware Speech Enhancement with Deep Complex U-Net", Hyeong-Seok Choi et al. https://arxiv.org/abs/1903.03107

masknet_class
alias of *asteroid.masknn.convolutional.DCUMaskNet*

class asteroid.models.demask.**DeMask**(*input_type='mag'*, *output_type='mag'*, *hidden_dims=(1024,)*, *dropout=0.0*, *activation='relu'*, *mask_act='relu'*, *norm_type='gLN'*, *fb_name='stft'*, *n_filters=512*, *stride=256*, *kernel_size=512*, *sample_rate=16000*, ***fb_kwargs*)
Bases: *asteroid.models.base_models.BaseEncoderMaskerDecoder*

Simple MLP model for surgical mask speech enhancement A transformed-domain masking approach is used.

Parameters

- **input_type** (*str, optional*) – whether the magnitude spectrogram "mag" or both real imaginary parts "reim" are passed as features to the masker network. Concatenation of "mag" and "reim" also can be used by using "cat".
- **output_type** (*str, optional*) – whether the masker ouputs a mask for magnitude spectrogram "mag" or both real imaginary parts "reim".
- **hidden_dims** (*list, optional*) – list of MLP hidden layer sizes.
- **dropout** (*float, optional*) – dropout probability.

- **activation** (*str, optional*) – type of activation used in hidden MLP layers.
- **mask_act** (*str, optional*) – Which non-linear function to generate mask.
- **norm_type** (*str, optional*) – To choose from `'BN'`, `'gLN'`, `'cLN'`.
- **fb_name** (*str*) – type of analysis and synthesis filterbanks used, choose between ["stft", "free", "analytic_free"].
- **n_filters** (*int*) – number of filters in the analysis and synthesis filterbanks.
- **stride** (*int*) – filterbank filters stride.
- **kernel_size** (*int*) – length of filters in the filterbank.
- **encoder_activation** (*str*) –
- **sample_rate** (*float*) – Sampling rate of the model.
- ****fb_kwargs** (*dict*) – Additional kwards to pass to the filterbank creation.

forward_masker(*tf_rep*)
Estimates masks based on time-frequency representations.

Parameters **tf_rep** (*torch.Tensor*) – Time-frequency representation in (batch, freq, seq).

Returns *torch.Tensor* – Estimated masks in (batch, freq, seq).

apply_masks(*tf_rep*, *est_masks*)
Applies masks to time-frequency representations.

Parameters

- **tf_rep** (*torch.Tensor*) – Time-frequency representations in (batch, freq, seq).
- **est_masks** (*torch.Tensor*) – Estimated masks in (batch, freq, seq).

Returns *torch.Tensor* – Masked time-frequency representations.

get_model_args()
Arguments needed to re-instantiate the model.

class asteroid.models.dprnn_tasnet.**DPRNNTasNet**(*n_src*, *out_chan=None*, *bn_chan=128*, *hid_size=128*, *chunk_size=100*, *hop_size=None*, *n_repeats=6*, *norm_type='gLN'*, *mask_act='sigmoid'*, *bidirectional=True*, *rnn_type='LSTM'*, *num_layers=1*, *dropout=0*, *in_chan=None*, *fb_name='free'*, *kernel_size=16*, *n_filters=64*, *stride=8*, *encoder_activation=None*, *sample_rate=8000*, *use_mulcat=False*, ***fb_kwargs*)
Bases: *asteroid.models.base_models.BaseEncoderMaskerDecoder*

DPRNN separation model, as described in [1].

Parameters

- **n_src** (*int*) – Number of masks to estimate.
- **out_chan** (*int or None*) – Number of bins in the estimated masks. Defaults to *in_chan*.
- **bn_chan** (*int*) – Number of channels after the bottleneck. Defaults to 128.

- **hid_size** (*int*) – Number of neurons in the RNNs cell state. Defaults to 128.
- **chunk_size** (*int*) – window size of overlap and add processing. Defaults to 100.
- **hop_size** (*int or None*) – hop size (stride) of overlap and add processing. Default to *chunk_size // 2* (50% overlap).
- **n_repeats** (*int*) – Number of repeats. Defaults to 6.
- **norm_type** (*str, optional*) – Type of normalization to use. To choose from
  - `'gLN'`: global Layernorm
  - `'cLN'`: channelwise Layernorm
- **mask_act** (*str, optional*) – Which non-linear function to generate mask.
- **bidirectional** (*bool, optional*) – True for bidirectional Inter-Chunk RNN (Intra-Chunk is always bidirectional).
- **rnn_type** (*str, optional*) – Type of RNN used. Choose between `'RNN'`, `'LSTM'` and `'GRU'`.
- **num_layers** (*int, optional*) – Number of layers in each RNN.
- **dropout** (*float, optional*) – Dropout ratio, must be in [0,1].
- **in_chan** (*int, optional*) – Number of input channels, should be equal to n_filters.
- **fb_name** (*str, className*) – Filterbank family from which to make encoder and decoder. To choose among `['free', 'analytic_free', 'param_sinc', 'stft']`.
- **n_filters** (*int*) – Number of filters / Input dimension of the masker net.
- **kernel_size** (*int*) – Length of the filters.
- **stride** (*int, optional*) – Stride of the convolution. If None (default), set to `kernel_size // 2`.
- **sample_rate** (*float*) – Sampling rate of the model.
- ****fb_kwargs** (*dict*) – Additional kwards to pass to the filterbank creation.

References

- [1] "Dual-path RNN: efficient long sequence modeling for time-domain single-channel speech separation", Yi Luo, Zhuo Chen and Takuya Yoshioka. https://arxiv.org/abs/1910.06379

class asteroid.models.dptnet.**DPTNet**(*n_src*, *n_heads=4*, *ff_hid=256*, *chunk_size=100*, *hop_size=None*, *n_repeats=6*, *norm_type='gLN'*, *ff_activation='relu'*, *encoder_activation='relu'*, *mask_act='relu'*, *bidirectional=True*, *dropout=0*, *in_chan=None*, *fb_name='free'*, *kernel_size=16*, *n_filters=64*, *stride=8*, *sample_rate=8000*, ***fb_kwargs*)

Bases: *asteroid.models.base_models.BaseEncoderMaskerDecoder*

DPTNet separation model, as described in [1].

Parameters

- **n_src** (*int*) – Number of masks to estimate.
- **out_chan** (*int or None*) – Number of bins in the estimated masks. Defaults to *in_chan*.

- **bn_chan** (*int*) – Number of channels after the bottleneck. Defaults to 128.
- **hid_size** (*int*) – Number of neurons in the RNNs cell state. Defaults to 128.
- **chunk_size** (*int*) – window size of overlap and add processing. Defaults to 100.
- **hop_size** (*int or None*) – hop size (stride) of overlap and add processing. Default to *chunk_size // 2* (50% overlap).
- **n_repeats** (*int*) – Number of repeats. Defaults to 6.
- **norm_type** (*str, optional*) – Type of normalization to use. To choose from
  - `'gLN'`: global Layernorm
  - `'cLN'`: channelwise Layernorm
- **mask_act** (*str, optional*) – Which non-linear function to generate mask.
- **bidirectional** (*bool, optional*) – True for bidirectional Inter-Chunk RNN (Intra-Chunk is always bidirectional).
- **rnn_type** (*str, optional*) – Type of RNN used. Choose between `'RNN'`, `'LSTM'` and `'GRU'`.
- **num_layers** (*int, optional*) – Number of layers in each RNN.
- **dropout** (*float, optional*) – Dropout ratio, must be in [0,1].
- **in_chan** (*int, optional*) – Number of input channels, should be equal to n_filters.
- **fb_name** (*str, className*) – Filterbank family from which to make encoder and decoder. To choose among `['free', 'analytic_free', 'param_sinc', 'stft']`.
- **n_filters** (*int*) – Number of filters / Input dimension of the masker net.
- **kernel_size** (*int*) – Length of the filters.
- **stride** (*int, optional*) – Stride of the convolution. If None (default), set to `kernel_size // 2`.
- **sample_rate** (*float*) – Sampling rate of the model.
- ****fb_kwargs** (*dict*) – Additional kwards to pass to the filterbank creation.

References

- [1]: Jingjing Chen et al. "Dual-Path Transformer Network: Direct Context-Aware Modeling for End-to-End Monaural Speech Separation" Interspeech 2020.

class asteroid.models.lstm_tasnet.**LSTMTasNet**(*n_src*, *out_chan=None*, *rnn_type='lstm'*, *n_layers=4*, *hid_size=512*, *dropout=0.3*, *mask_act='sigmoid'*, *bidirectional=True*, *in_chan=None*, *fb_name='free'*, *n_filters=64*, *kernel_size=16*, *stride=8*, *encoder_activation=None*, *sample_rate=8000*, ***fb_kwargs*)

Bases: *asteroid.models.base_models.BaseEncoderMaskerDecoder*

TasNet separation model, as described in [1].

Parameters

- **n_src** (*int*) – Number of masks to estimate.

- **out_chan** (*int or None*) – Number of bins in the estimated masks. Defaults to *in_chan*.
- **hid_size** (*int*) – Number of neurons in the RNNs cell state. Defaults to 128.
- **mask_act** (*str, optional*) – Which non-linear function to generate mask.
- **bidirectional** (*bool, optional*) – True for bidirectional Inter-Chunk RNN (Intra-Chunk is always bidirectional).
- **rnn_type** (*str, optional*) – Type of RNN used. Choose between `'RNN'`, `'LSTM'` and `'GRU'`.
- **n_layers** (*int, optional*) – Number of layers in each RNN.
- **dropout** (*float, optional*) – Dropout ratio, must be in [0,1].
- **in_chan** (*int, optional*) – Number of input channels, should be equal to n_filters.
- **fb_name** (*str, className*) – Filterbank family from which to make encoder and decoder. To choose among [`'free'`, `'analytic_free'`, `'param_sinc'`, `'stft'`].
- **n_filters** (*int*) – Number of filters / Input dimension of the masker net.
- **kernel_size** (*int*) – Length of the filters.
- **stride** (*int, optional*) – Stride of the convolution. If None (default), set to `kernel_size // 2`.
- **sample_rate** (*float*) – Sampling rate of the model.
- ****fb_kwargs** (*dict*) – Additional kwards to pass to the filterbank creation.

References

- [1]: Yi Luo et al. "Real-time Single-channel Dereverberation and Separation with Time-domain Audio Separation Network", Interspeech 2018

class asteroid.models.sudormrf.**SuDORMRFNet**(*n_src*, *bn_chan=128*, *num_blocks=16*, *upsampling_depth=4*, *mask_act='softmax'*, *in_chan=None*, *fb_name='free'*, *kernel_size=21*, *n_filters=512*, *stride=None*, *sample_rate=8000*, ***fb_kwargs*)

Bases: *asteroid.models.base_models.BaseEncoderMaskerDecoder*

SuDORMRF separation model, as described in [1].

Parameters

- **n_src** (*int*) – Number of sources in the input mixtures.
- **bn_chan** (*int, optional*) – Number of bins in the bottleneck layer and the UNet blocks.
- **num_blocks** (*int*) – Number of of UBlocks.
- **upsampling_depth** (*int*) – Depth of upsampling.
- **mask_act** (*str*) – Name of output activation.
- **in_chan** (*int, optional*) – Number of input channels, should be equal to n_filters.
- **fb_name** (*str, className*) – Filterbank family from which to make encoder and decoder. To choose among [`'free'`, `'analytic_free'`, `'param_sinc'`, `'stft'`].
- **n_filters** (*int*) – Number of filters / Input dimension of the masker net.

- **kernel_size** (*int*) – Length of the filters.
- **stride** (*int, optional*) – Stride of the convolution. If None (default), set to `kernel_size // 2`.
- **sample_rate** (*float*) – Sampling rate of the model.
- ****fb_kwargs** (*dict*) – Additional kwards to pass to the filterbank creation.

References

- [1] : "Sudo rm -rf: Efficient Networks for Universal Audio Source Separation", Tzinis et al. MLSP 2020.

class asteroid.models.sudormrf.**SuDORMRFImprovedNet**(*n_src*, *bn_chan=128*, *num_blocks=16*, *upsampling_depth=4*, *mask_act='relu'*, *in_chan=None*, *fb_name='free'*, *kernel_size=21*, *n_filters=512*, *stride=None*, *sample_rate=8000*, ***fb_kwargs*)

Bases: *asteroid.models.base_models.BaseEncoderMaskerDecoder*

Improved SuDORMRF separation model, as described in [1].

Parameters

- **n_src** (*int*) – Number of sources in the input mixtures.
- **bn_chan** (*int, optional*) – Number of bins in the bottleneck layer and the UNet blocks.
- **num_blocks** (*int*) – Number of of UBlocks.
- **upsampling_depth** (*int*) – Depth of upsampling.
- **mask_act** (*str*) – Name of output activation.
- **in_chan** (*int, optional*) – Number of input channels, should be equal to n_filters.
- **fb_name** (*str, className*) – Filterbank family from which to make encoder and decoder. To choose among `['free', 'analytic_free', 'param_sinc', 'stft']`.
- **n_filters** (*int*) – Number of filters / Input dimension of the masker net.
- **kernel_size** (*int*) – Length of the filters.
- **stride** (*int, optional*) – Stride of the convolution. If None (default), set to `kernel_size // 2`.
- ****fb_kwargs** (*dict*) – Additional kwards to pass to the filterbank creation.

References

- [1] : "Sudo rm -rf: Efficient Networks for Universal Audio Source Separation", Tzinis et al. MLSP 2020.

12.3 Publishing models

class asteroid.models.zenodo.**Zenodo**(*api_key=None*, *use_sandbox=True*)

Bases: object

Faciliate Zenodo's REST API.

Parameters

- **api_key** (*str*) – Access token generated to upload depositions.
- **use_sandbox** (*bool*) – Whether to use the sandbox (default: True) Note that *api_key* are different in sandbox.

All methods return the requests response.

Note: A Zenodo record is something that is public and cannot be deleted. A Zenodo deposit has not yet been published, is private and can be deleted.

---

create_new_deposition (*metadata=None*)
Creates a new deposition.

Parameters **metadata** (*dict, optional*) – Metadata dict to upload on the new deposition.

change_metadata_in_deposition (*dep_id*, *metadata*)
Set or replace metadata in given deposition

Parameters

- **dep_id** (*int*) – deposition id. You cna get it with *r = create_new_deposition(); dep_id = r.json()['id']*
- **metadata** (*dict*) – Metadata dict.

Examples

```
>>> metadata = {
...     'title': 'My first upload',
...     'upload_type': 'poster',
...     'description': 'This is my first upload',
...     'creators': [{'name': 'Doe, John',
...                   'affiliation': 'Zenodo'}]
... }
```

upload_new_file_to_deposition (*dep_id*, *file*, *name=None*)
Upload one file to existing deposition.

Parameters

- **dep_id** (*int*) – deposition id. You cna get it with *r = create_new_deposition(); dep_id = r.json()['id']*
- **file** (*str or io.BufferedReader*) – path to a file, or already opened file (path prefered).
- **name** (*str, optional*) – name given to the uploaded file. Defaults to the path.

(More: https://developers.zenodo.org/#deposition-files)

publish_deposition (*dep_id*)
Publish given deposition (Cannot be deleted)!

Parameters **dep_id** (*int*) – deposition id. You cna get it with *r = create_new_deposition(); dep_id = r.json()['id']*

get_deposition(*dep_id=-1*)
Get deposition by deposition id. Get all dep_id is -1 (default).

remove_deposition(*dep_id*)
Remove deposition with deposition id *dep_id*

remove_all_depositions()
Removes all unpublished deposition (not records).

asteroid.models.publisher.**save_publishable**(*publish_dir*, *model_dict*, *metrics=None*, *train_conf=None*, *recipe=None*)
Save models to prepare for publication / model sharing.

Parameters

- **publish_dir** (*str*) – Path to the publishing directory. Usually under exp/exp_name/publish_dir
- **model_dict** (*dict*) – dict at least with keys *model_args*, *state_dict*,'dataset' or *licenses*
- **metrics** (*dict*) – dict with evaluation metrics.
- **train_conf** (*dict*) – Training configuration dict (from conf.yml).
- **recipe** (*str*) – Name of the recipe.

Returns dict, same as *model_dict* with added fields.

Raises *AssertionError when either 'model_args', 'state_dict','dataset' or – licenses* are not present is *model_dict.keys()*

asteroid.models.publisher.**upload_publishable**(*publish_dir*, *uploader=None*, *affiliation=None*, *git_username=None*, *token=None*, *force_publish=False*, *use_sandbox=False*, *unit_test=False*)
Entry point to upload publishable model.

Parameters

- **publish_dir** (*str*) – Path to the publishing directory. Usually under exp/exp_name/publish_dir
- **uploader** (*str*) – Full name of the uploader (Ex: Manuel Pariente)
- **affiliation** (*str, optional*) – Affiliation (no accent).
- **git_username** (*str, optional*) – GitHub username.
- **token** (*str*) – Access token generated to upload depositions.
- **force_publish** (*bool*) – Whether to directly publish without asking confirmation before. Defaults to False.
- **use_sandbox** (*bool*) – Whether to use Zenodo's sandbox instead of the official Zenodo.
- **unit_test** (*bool*) – If True, we do not ask user input and do not publish.

asteroid.models.publisher.**get_username**()
Get git of FS username for upload.

asteroid.models.publisher.**make_license_notice**(*model_name*, *licenses*, *uploader=None*)
Make license notice based on license dicts.

Parameters

- **model_name** (*str*) – Name of the model.

- **licenses** (*List[dict]*) – List of dict with keys (*title*, *title_link*, *author*, *author_link*, *licence*, *licence_link*).
- **uploader** (*str*) – Name of the uploader such as "Manuel Pariente".

Returns

str, the license note describing the model, it's attribution, the original licenses, what we license it under and the licensor.

asteroid.models.publisher.**zenodo_upload**(*model*, *token*, *model_path=None*, *use_sandbox=False*)

Create deposit and upload metadata + model

Parameters

- **model** (*dict*) –
- **token** (*str*) – Access token.
- **model_path** (*str*) – Saved model path.
- **use_sandbox** (*bool*) – Whether to use Zenodo's sandbox instead of the official Zenodo.

Returns Zenodo (Zenodo instance with access token) int (deposit ID)

asteroid.models.publisher.**make_metadata_from_model**(*model*)

Create Zenodo deposit metadata for a given publishable model.

Parameters **model** (*dict*) – Dictionary with all infos needed to publish. More info to come.

Returns dict, the metadata to create the Zenodo deposit with.

Note: We remove the PESQ from the final results as a license is needed to use it.

---

asteroid.models.publisher.**two_level_dict_html**(*dic*)

Two-level dict to HTML.

Parameters **dic** (*dict*) – two-level dict

Returns str for HTML-encoded two level dic

asteroid.models.publisher.**display_one_level_dict**(*dic*)

Single level dict to HTML

Parameters **dic** (*dict*) –

Returns str for HTML-encoded single level dic

CHAPTER 13

Losses & Metrics

13.1 Permutation invariant training (PIT) made easy

Asteroid supports regular Permutation Invariant Training (PIT), it's extension using Sinkhorn algorithm (SinkPIT) as well as Mixture Invariant Training (MixIT).

13.1.1 PIT

class asteroid.losses.pit_wrapper.**PITLossWrapper**(*loss_func*, *pit_from='pw_mtx'*, *perm_reduce=None*)

Bases: sphinx.ext.autodoc.importer._MockObject

Permutation invariant loss wrapper.

Parameters

- **loss_func** – function with signature (est_targets, targets, **kwargs).
- **pit_from** (*str*) – Determines how PIT is applied.
  - `'pw_mtx'` (pairwise matrix): *loss_func* computes pairwise losses and returns a torch.Tensor of shape $(batch, n\_src, n\_src)$. Each element $(batch, i, j)$ corresponds to the loss between $targets[:, i]$ and $est\_targets[:, j]$
  - `'pw_pt'` (pairwise point): *loss_func* computes the loss for a batch of single source and single estimates (tensors won't have the source axis). Output shape : $(batch)$. See *get_pw_losses()*.
  - `'perm_avg'` (permutation average): *loss_func* computes the average loss for a given permutations of the sources and estimates. Output shape : $(batch)$. See *best_perm_from_perm_avg_loss()*.

  In terms of efficiency, `'perm_avg'` is the least efficicient.
- **perm_reduce** (*Callable*) – torch function to reduce permutation losses. Defaults to None (equivalent to mean). Signature of the func (pwl_set, **kwargs) :

$(B, n\_src!, n\_src) -- > (B, n\_src!)$. *perm_reduce* can receive **kwargs during forward using the *reduce_kwargs* argument (dict). If those argument are static, consider defining a small function or using *functools.partial*. Only used in *'pw_mtx'* and *'pw_pt' pit_from* modes.

For each of these modes, the best permutation and reordering will be automatically computed. When either `'pw_mtx'` or `'pw_pt'` is used, and the number of sources is larger than three, the hungarian algorithm is used to find the best permutation.

Examples

```
>>> import torch
>>> from asteroid.losses import pairwise_neg_sisdr
>>> sources = torch.randn(10, 3, 16000)
>>> est_sources = torch.randn(10, 3, 16000)
>>> # Compute PIT loss based on pairwise losses
>>> loss_func = PITLossWrapper(pairwise_neg_sisdr, pit_from='pw_mtx')
>>> loss_val = loss_func(est_sources, sources)
>>>
>>> # Using reduce
>>> def reduce(perm_loss, src):
>>>     weighted = perm_loss * src.norm(dim=-1, keepdim=True)
>>>     return torch.mean(weighted, dim=-1)
>>>
>>> loss_func = PITLossWrapper(pairwise_neg_sisdr, pit_from='pw_mtx',
>>>                            perm_reduce=reduce)
>>> reduce_kwargs = {'src': sources}
>>> loss_val = loss_func(est_sources, sources,
>>>                      reduce_kwargs=reduce_kwargs)
```

forward (*est_targets*, *targets*, *return_est=False*, *reduce_kwargs=None*, ***kwargs*)

Find the best permutation and return the loss.

Parameters

- **est_targets** – torch.Tensor. Expected shape $(batch, nsrc, ...)$. The batch of target estimates.
- **targets** – torch.Tensor. Expected shape $(batch, nsrc, ...)$. The batch of training targets
- **return_est** – Boolean. Whether to return the reordered targets estimates (To compute metrics or to save example).
- **reduce_kwargs** (*dict or None*) – kwargs that will be passed to the pairwise losses reduce function (*perm_reduce*).
- ****kwargs** – additional keyword argument that will be passed to the loss function.

Returns

- Best permutation loss for each batch sample, average over the batch.
- The reordered targets estimates if `return_est` is True. `torch.Tensor` of shape $(batch, nsrc, ...)$.

static get_pw_losses (*loss_func*, *est_targets*, *targets*, ***kwargs*)

Get pair-wise losses between the training targets and its estimate for a given loss function.

Parameters

- **loss_func** – function with signature (est_targets, targets, **kwargs) The loss function to get pair-wise losses from.

- **est_targets** – torch.Tensor. Expected shape $(batch, nsrc, ...)$. The batch of target estimates.
- **targets** – torch.Tensor. Expected shape $(batch, nsrc, ...)$. The batch of training targets.
- ****kwargs** – additional keyword argument that will be passed to the loss function.

Returns torch.Tensor or size $(batch, nsrc, nsrc)$, losses computed for all permutations of the targets and est_targets.

This function can be called on a loss function which returns a tensor of size $(batch)$. There are more efficient ways to compute pair-wise losses using broadcasting.

static best_perm_from_perm_avg_loss(*loss_func*, *est_targets*, *targets*, ***kwargs*)
Find best permutation from loss function with source axis.

Parameters

- **loss_func** – function with signature $(est_targets, targets, **kwargs)$ The loss function batch losses from.
- **est_targets** – torch.Tensor. Expected shape $(batch, nsrc, *)$. The batch of target estimates.
- **targets** – torch.Tensor. Expected shape $(batch, nsrc, *)$. The batch of training targets.
- ****kwargs** – additional keyword argument that will be passed to the loss function.

Returns

- `torch.Tensor` – The loss corresponding to the best permutation of size $(batch,)$.
- **torch.Tensor:** The indices of the best permutations.

static find_best_perm(*pair_wise_losses*, *perm_reduce=None*, ***kwargs*)
Find the best permutation, given the pair-wise losses.

Dispatch between factorial method if number of sources is small (<3) and hungarian method for more sources. If `perm_reduce` is not None, the factorial method is always used.

Parameters

- **pair_wise_losses** (`torch.Tensor`) – Tensor of shape $(batch, n\_src, n\_src)$. Pairwise losses.
- **perm_reduce** (*Callable*) – torch function to reduce permutation losses. Defaults to None (equivalent to mean). Signature of the func (pwl_set, **kwargs) : $(B, n\_src!, n\_src) -> (B, n\_src!)$
- ****kwargs** – additional keyword argument that will be passed to the permutation reduce function.

Returns

- `torch.Tensor` – The loss corresponding to the best permutation of size $(batch,)$.
- `torch.Tensor`: The indices of the best permutations.

static reorder_source(*source*, *batch_indices*)
Reorder sources according to the best permutation.

Parameters

- **source** (*torch.Tensor*) – Tensor of shape $(batch, n_src, time)$

- **batch_indices** (*torch.Tensor*) – Tensor of shape $(batch, n_src)$. Contains optimal permutation indices for each batch.

Returns torch.Tensor – Reordered sources.

static find_best_perm_factorial(*pair_wise_losses*, *perm_reduce=None*, ***kwargs*)

Find the best permutation given the pair-wise losses by looping through all the permutations.

Parameters

- **pair_wise_losses** (torch.Tensor) – Tensor of shape $(batch, n_src, n_src)$. Pairwise losses.
- **perm_reduce** (*Callable*) – torch function to reduce permutation losses. Defaults to None (equivalent to mean). Signature of the func (pwl_set, **kwargs) : $(B, n\_src!, n\_src) -> (B, n\_src!)$
- ****kwargs** – additional keyword argument that will be passed to the permutation reduce function.

Returns

- torch.Tensor – The loss corresponding to the best permutation of size $(batch,)$.
- torch.Tensor: The indices of the best permutations.

MIT Copyright (c) 2018 Kaituo XU. See Original code and License.

static find_best_perm_hungarian(*pair_wise_losses: <sphinx.ext.autodoc.importer._MockObject object at 0x7f96ea100b10>*)

Find the best permutation given the pair-wise losses, using the Hungarian algorithm.

Returns

- torch.Tensor – The loss corresponding to the best permutation of size (batch,).
- torch.Tensor: The indices of the best permutations.

class asteroid.losses.pit_wrapper.**PITReorder**(*loss_func*, *pit_from='pw_mtx'*, *perm_reduce=None*)

Bases: *asteroid.losses.pit_wrapper.PITLossWrapper*

Permutation invariant reorderer. Only returns the reordered estimates. See *:py:class:asteroid.losses.PITLossWrapper*.

forward(*est_targets*, *targets*, *reduce_kwargs=None*, ***kwargs*)

Find the best permutation and return the loss.

Parameters

- **est_targets** – torch.Tensor. Expected shape $(batch, nsrc, ...)$. The batch of target estimates.
- **targets** – torch.Tensor. Expected shape $(batch, nsrc, ...)$. The batch of training targets
- **return_est** – Boolean. Whether to return the reordered targets estimates (To compute metrics or to save example).
- **reduce_kwargs** (*dict or None*) – kwargs that will be passed to the pairwise losses reduce function (*perm_reduce*).
- ****kwargs** – additional keyword argument that will be passed to the loss function.

Returns

- Best permutation loss for each batch sample, average over the batch.

- The reordered targets estimates if `return_est` is True. `torch.Tensor` of shape $(batch, nsrc, ...)$.

13.1.2 MixIT

class asteroid.losses.mixit_wrapper.**MixITLossWrapper**(*loss_func*, *generalized=True*, *reduction='mean'*)

Bases: `sphinx.ext.autodoc.importer._MockObject`

Mixture invariant loss wrapper.

Parameters

- **loss_func** – function with signature (est_targets, targets, **kwargs).
- **generalized** (*bool*) – Determines how MixIT is applied. If False , apply MixIT for any number of mixtures as soon as they contain the same number of sources (*best_part_mixit()*.) If True (default), apply MixIT for two mixtures, but those mixtures do not necessarily have to contain the same number of sources. See *best_part_mixit_generalized()*.
- **reduction** (*string, optional*) – Specifies the reduction to apply to the output: `'none' | 'mean'`. `'none'`: no reduction will be applied, `'mean'`: the sum of the output will be divided by the number of elements in the output.

For each of these modes, the best partition and reordering will be automatically computed.

Examples

```
>>> import torch
>>> from asteroid.losses import multisrc_mse
>>> mixtures = torch.randn(10, 2, 16000)
>>> est_sources = torch.randn(10, 4, 16000)
>>> # Compute MixIT loss based on pairwise losses
>>> loss_func = MixITLossWrapper(multisrc_mse)
>>> loss_val = loss_func(est_sources, mixtures)
```

References [1] Scott Wisdom et al. "Unsupervised sound separation using mixtures of mixtures." arXiv:2006.12701 (2020)

forward(*est_targets*, *targets*, *return_est=False*, ***kwargs*)

Find the best partition and return the loss.

Parameters

- **est_targets** – torch.Tensor. Expected shape $(batch, nsrc, *)$. The batch of target estimates.
- **targets** – torch.Tensor. Expected shape $(batch, nmix, ...)$. The batch of training targets
- **return_est** – Boolean. Whether to return the estimated mixtures estimates (To compute metrics or to save example).
- ****kwargs** – additional keyword argument that will be passed to the loss function.

Returns

- Best partition loss for each batch sample, average over the batch. torch.Tensor(loss_value)

- The estimated mixtures (estimated sources summed according to the partition) if return_est is True. torch.Tensor of shape $(batch, nmix, ...)$.

static best_part_mixit(*loss_func*, *est_targets*, *targets*, ***kwargs*)

Find best partition of the estimated sources that gives the minimum loss for the MixIT training paradigm in [1]. Valid for any number of mixtures as soon as they contain the same number of sources.

Parameters

- **loss_func** – function with signature `(est_targets, targets, **kwargs)` The loss function to get batch losses from.
- **est_targets** – torch.Tensor. Expected shape $(batch, nsrc, ...)$. The batch of target estimates.
- **targets** – torch.Tensor. Expected shape $(batch, nmix, ...)$. The batch of training targets (mixtures).
- ****kwargs** – additional keyword argument that will be passed to the loss function.

Returns

- `torch.Tensor` – The loss corresponding to the best permutation of size (batch,).
- `torch.LongTensor`: The indices of the best partition.
- `list`: list of the possible partitions of the sources.

static best_part_mixit_generalized(*loss_func*, *est_targets*, *targets*, ***kwargs*)

Find best partition of the estimated sources that gives the minimum loss for the MixIT training paradigm in [1]. Valid only for two mixtures, but those mixtures do not necessarly have to contain the same number of sources e.g the case where one mixture is silent is allowed..

Parameters

- **loss_func** – function with signature `(est_targets, targets, **kwargs)` The loss function to get batch losses from.
- **est_targets** – torch.Tensor. Expected shape $(batch, nsrc, ...)$. The batch of target estimates.
- **targets** – torch.Tensor. Expected shape $(batch, nmix, ...)$. The batch of training targets (mixtures).
- ****kwargs** – additional keyword argument that will be passed to the loss function.

Returns

- `torch.Tensor` – The loss corresponding to the best permutation of size (batch,).
- `torch.LongTensor`: The indexes of the best permutations.
- `list`: list of the possible partitions of the sources.

static loss_set_from_parts(*loss_func*, *est_targets*, *targets*, *parts*, ***kwargs*)

Common loop between both best_part_mixit

static reorder_source(*est_targets*, *targets*, *min_loss_idx*, *parts*)

Reorder sources according to the best partition.

Parameters

- **est_targets** – torch.Tensor. Expected shape $(batch, nsrc, ...)$. The batch of target estimates.

- **targets** – torch.Tensor. Expected shape $(batch, nmix, ...)$. The batch of training targets.
- **min_loss_idx** – torch.LongTensor. The indexes of the best permutations.
- **parts** – list of the possible partitions of the sources.

Returns `torch.Tensor` – Reordered sources of shape $(batch, nmix, time)$.

13.1.3 SinkPIT

class asteroid.losses.sinkpit_wrapper.**SinkPITLossWrapper**(*loss_func*, *n_iter=200*, *hungarian_validation=True*)

Bases: `sphinx.ext.autodoc.importer._MockObject`

Permutation invariant loss wrapper.

Parameters

- **loss_func** – function with signature (targets, est_targets, **kwargs).
- **n_iter** (*int*) – number of the Sinkhorn iteration (default = 200). Supposed to be an even number.
- **hungarian_validation** (*boolean*) – Whether to use the Hungarian algorithm for the validation. (default = True)

`loss_func` computes pairwise losses and returns a torch.Tensor of shape $(batch, n\_src, n\_src)$. Each element $(batch, i, j)$ corresponds to the loss between $targets[:, i]$ and $est\_targets[:, j]$ It evaluates an approximate value of the PIT loss using Sinkhorn's iterative algorithm. See `best_softperm_sinkhorn()` and http://arxiv.org/abs/2010.11871

Examples

```
>>> import torch
>>> import pytorch_lightning as pl
>>> from asteroid.losses import pairwise_neg_sisdr
>>> sources = torch.randn(10, 3, 16000)
>>> est_sources = torch.randn(10, 3, 16000)
>>> # Compute SinkPIT loss based on pairwise losses
>>> loss_func = SinkPITLossWrapper(pairwise_neg_sisdr)
>>> loss_val = loss_func(est_sources, sources)
>>> # A fixed temperature parameter `beta` (=10) is used
>>> # unless a cooling callback is set. The value can be
>>> # dynamically changed using a cooling callback module as follows.
>>> model = NeuralNetworkModel()
>>> optimizer = optim.Adam(model.parameters(), lr=1e-3)
>>> dataset = YourDataset()
>>> loader = data.DataLoader(dataset, batch_size=16)
>>> system = System(
>>>     model,
>>>     optimizer,
>>>     loss_func=SinkPITLossWrapper(pairwise_neg_sisdr),
>>>     train_loader=loader,
>>>     val_loader=loader,
>>>     )
>>>
>>> trainer = pl.Trainer(
>>>     max_epochs=100,
```

(continues on next page)

(continued from previous page)

```
>>>     callbacks=[SinkPITBetaScheduler(lambda epoch : 1.02 ** epoch)],
>>>     )
>>>
>>> trainer.fit(system)
```

forward(*est_targets*, *targets*, *return_est=False*, ***kwargs*)
Evaluate the loss using Sinkhorn's algorithm.

Parameters

- **est_targets** – torch.Tensor. Expected shape $(batch, nsrc, ...)$. The batch of target estimates.
- **targets** – torch.Tensor. Expected shape $(batch, nsrc, ...)$. The batch of training targets
- **return_est** – Boolean. Whether to return the reordered targets estimates (To compute metrics or to save example).
- ****kwargs** – additional keyword argument that will be passed to the loss function.

Returns

- **Best permutation loss for each batch sample, average over** the batch. torch.Tensor(loss_value)
- **The reordered targets estimates if return_est is True.** torch.Tensor of shape $(batch, nsrc, ...)$.

static best_softperm_sinkhorn(*pair_wise_losses*, *beta=10*, *n_iter=200*)
Compute an approximate PIT loss using Sinkhorn's algorithm. See http://arxiv.org/abs/2010.11871

Parameters

- **pair_wise_losses** (*torch.Tensor*) – Tensor of shape $(batch, n_src, n_src)$. Pairwise losses.
- **beta** (*float*) – Inverse temperature parameter. (default = 10)
- **n_iter** (*int*) – Number of iteration. Even number. (default = 200)

Returns

- *torch.Tensor* – The loss corresponding to the best permutation of size (batch,).
- *torch.Tensor*: A soft permutation matrix.

13.2 Available loss functions

`PITLossWrapper` supports three types of loss function. For "easy" losses, we implement the three types (pairwise point, single-source loss and multi-source loss). For others, we only implement the single-source loss which can be aggregated into both PIT and nonPIT training.

13.2.1 MSE

asteroid.losses.mse.**PairwiseMSE**(**args*, ***kwargs*)
Measure pairwise mean square error on a batch.

Shape:

- est_targets : $(batch, nsrc, ...)$.
- targets: $(batch, nsrc, ...)$.

Returns `torch.Tensor` – with shape $(batch, nsrc, nsrc)$

Examples

```
>>> import torch
>>> from asteroid.losses import PITLossWrapper
>>> targets = torch.randn(10, 2, 32000)
>>> est_targets = torch.randn(10, 2, 32000)
>>> loss_func = PITLossWrapper(PairwiseMSE(), pit_from='pairwise')
>>> loss = loss_func(est_targets, targets)
```

asteroid.losses.mse.**SingleSrcMSE**(**args*, ***kwargs*)

Measure mean square error on a batch. Supports both tensors with and without source axis.

Shape:

- est_targets: $(batch, ...)$.
- targets: $(batch, ...)$.

Returns `torch.Tensor` – with shape $(batch)$

Examples

```
>>> import torch
>>> from asteroid.losses import PITLossWrapper
>>> targets = torch.randn(10, 2, 32000)
>>> est_targets = torch.randn(10, 2, 32000)
>>> # singlesrc_mse / multisrc_mse support both 'pw_pt' and 'perm_avg'.
>>> loss_func = PITLossWrapper(singlesrc_mse, pit_from='pw_pt')
>>> loss = loss_func(est_targets, targets)
```

asteroid.losses.mse.**MultiSrcMSE**(**args*, ***kwargs*)

Measure mean square error on a batch. Supports both tensors with and without source axis.

Shape:

- est_targets: $(batch, ...)$.
- targets: $(batch, ...)$.

Returns `torch.Tensor` – with shape $(batch)$

Examples

```
>>> import torch
>>> from asteroid.losses import PITLossWrapper
>>> targets = torch.randn(10, 2, 32000)
>>> est_targets = torch.randn(10, 2, 32000)
>>> # singlesrc_mse / multisrc_mse support both 'pw_pt' and 'perm_avg'.
>>> loss_func = PITLossWrapper(singlesrc_mse, pit_from='pw_pt')
>>> loss = loss_func(est_targets, targets)
```

13.2.2 SDR

asteroid.losses.sdr.**PairwiseNegSDR**(**args*, ***kwargs*)

Base class for pairwise negative SI-SDR, SD-SDR and SNR on a batch.

Parameters

- **sdr_type** (*str*) – choose between `snr` for plain SNR, `sisdr` for SI-SDR and `sdsdr` for SD-SDR [1].
- **zero_mean** (*bool, optional*) – by default it zero mean the target and estimate before computing the loss.
- **take_log** (*bool, optional*) – by default the log10 of sdr is returned.

Shape:

- est_targets : $(batch, nsrc, ...)$.
- targets: $(batch, nsrc, ...)$.

Returns `torch.Tensor` – with shape $(batch, nsrc, nsrc)$. Pairwise losses.

Examples

```
>>> import torch
>>> from asteroid.losses import PITLossWrapper
>>> targets = torch.randn(10, 2, 32000)
>>> est_targets = torch.randn(10, 2, 32000)
>>> loss_func = PITLossWrapper(PairwiseNegSDR("sisdr"),
>>>                            pit_from='pairwise')
>>> loss = loss_func(est_targets, targets)
```

References [1] Le Roux, Jonathan, et al. "SDR half-baked or well done." IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2019.

asteroid.losses.sdr.**SingleSrcNegSDR**(**args*, ***kwargs*)

Base class for single-source negative SI-SDR, SD-SDR and SNR.

Parameters

- **sdr_type** (*str*) – choose between `snr` for plain SNR, `sisdr` for SI-SDR and `sdsdr` for SD-SDR [1].
- **zero_mean** (*bool, optional*) – by default it zero mean the target and estimate before computing the loss.
- **take_log** (*bool, optional*) – by default the log10 of sdr is returned.
- **reduction** (*string, optional*) – Specifies the reduction to apply to the output: `'none'` | `'mean'`. `'none'`: no reduction will be applied, `'mean'`: the sum of the output will be divided by the number of elements in the output.

Shape:

- est_targets : $(batch, time)$.
- targets: $(batch, time)$.

Returns `torch.Tensor` – with shape $(batch)$ if `reduction='none'` else [] scalar if `reduction='mean'`.

Examples

```
>>> import torch
>>> from asteroid.losses import PITLossWrapper
>>> targets = torch.randn(10, 2, 32000)
>>> est_targets = torch.randn(10, 2, 32000)
>>> loss_func = PITLossWrapper(SingleSrcNegSDR("sisdr"),
>>>                            pit_from='pw_pt')
>>> loss = loss_func(est_targets, targets)
```

References [1] Le Roux, Jonathan, et al. "SDR half-baked or well done." IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2019.

asteroid.losses.sdr.**MultiSrcNegSDR**(**args*, ***kwargs*)

Base class for computing negative SI-SDR, SD-SDR and SNR for a given permutation of source and their estimates.

Parameters

- **sdr_type** (*str*) – choose between `snr` for plain SNR, `sisdr` for SI-SDR and `sdsdr` for SD-SDR [1].
- **zero_mean** (*bool, optional*) – by default it zero mean the target and estimate before computing the loss.
- **take_log** (*bool, optional*) – by default the log10 of sdr is returned.

Shape:

- est_targets : $(batch, nsrc, time)$.
- targets: $(batch, nsrc, time)$.

Returns `torch.Tensor` – with shape $(batch)$ if `reduction='none'` else [] scalar if `reduction='mean'`.

Examples

```
>>> import torch
>>> from asteroid.losses import PITLossWrapper
>>> targets = torch.randn(10, 2, 32000)
>>> est_targets = torch.randn(10, 2, 32000)
>>> loss_func = PITLossWrapper(MultiSrcNegSDR("sisdr"),
>>>                            pit_from='perm_avg')
>>> loss = loss_func(est_targets, targets)
```

References [1] Le Roux, Jonathan, et al. "SDR half-baked or well done." IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2019.

13.2.3 PMSQE

asteroid.losses.pmsqe.**SingleSrcPMSQE**(**args*, ***kwargs*)

Computes the Perceptual Metric for Speech Quality Evaluation (PMSQE) as described in [1]. This version is only designed for 16 kHz (512 length DFT). Adaptation to 8 kHz could be done by changing the parameters of the class (see Tensorflow implementation). The SLL, frequency and gain equalization are applied in each sequence independently.

Parameters

- **window_name** (*str*) – Select the used window function for the correct factor to be applied. Defaults to sqrt hanning window. Among ['rect', 'hann', 'sqrt_hann', 'hamming', 'flatTop'].
- **window_weight** (*float, optional*) – Correction to the window factor applied.
- **bark_eq** (*bool, optional*) – Whether to apply bark equalization.
- **gain_eq** (*bool, optional*) – Whether to apply gain equalization.
- **sample_rate** (*int*) – Sample rate of the input audio.

References [1] J.M.Martin, A.M.Gomez, J.A.Gonzalez, A.M.Peinado 'A Deep Learning Loss Function based on the Perceptual Evaluation of the Speech Quality', IEEE Signal Processing Letters, 2018. Implemented by Juan M. Martin. Contact: mdjuamart@ugr.es

---

Note: Inspired on the Perceptual Evaluation of the Speech Quality (PESQ) algorithm, this function consists of two regularization factors : the symmetrical and asymmetrical distortion in the loudness domain.

Examples

```
>>> import torch
>>> from asteroid_filterbanks import STFTFB, Encoder, transforms
>>> from asteroid.losses import PITLossWrapper, SingleSrcPMSQE
>>> stft = Encoder(STFTFB(kernel_size=512, n_filters=512, stride=256))
>>> # Usage by itself
>>> ref, est = torch.randn(2, 1, 16000), torch.randn(2, 1, 16000)
>>> ref_spec = transforms.mag(stft(ref))
>>> est_spec = transforms.mag(stft(est))
>>> loss_func = SingleSrcPMSQE()
>>> loss_value = loss_func(est_spec, ref_spec)
>>> # Usage with PITLossWrapper
>>> loss_func = PITLossWrapper(SingleSrcPMSQE(), pit_from='pw_pt')
>>> ref, est = torch.randn(2, 3, 16000), torch.randn(2, 3, 16000)
>>> ref_spec = transforms.mag(stft(ref))
>>> est_spec = transforms.mag(stft(est))
>>> loss_value = loss_func(ref_spec, est_spec)
```

13.2.4 STOI

13.2.5 MultiScale Spectral Loss

asteroid.losses.multi_scale_spectral.**SingleSrcMultiScaleSpectral**(**args*, ***kwargs*)

Measure multi-scale spectral loss as described in [1]

Parameters

- **n_filters** (*list*) – list containing the number of filter desired for each STFT
- **windows_size** (*list*) – list containing the size of the window desired for each STFT
- **hops_size** (*list*) – list containing the size of the hop desired for each STFT

Shape:

- est_targets : $(batch, time)$.
- targets: $(batch, time)$.

Returns `torch.Tensor` – with shape [batch]

Examples

```
>>> import torch
>>> targets = torch.randn(10, 32000)
>>> est_targets = torch.randn(10, 32000)
>>> # Using it by itself on a pair of source/estimate
>>> loss_func = SingleSrcMultiScaleSpectral()
>>> loss = loss_func(est_targets, targets)
```

```
>>> import torch
>>> from asteroid.losses import PITLossWrapper
>>> targets = torch.randn(10, 2, 32000)
>>> est_targets = torch.randn(10, 2, 32000)
>>> # Using it with PITLossWrapper with sets of source/estimates
>>> loss_func = PITLossWrapper(SingleSrcMultiScaleSpectral(),
>>>                            pit_from='pw_pt')
>>> loss = loss_func(est_targets, targets)
```

References [1] Jesse Engel and Lamtharn (Hanoi) Hantrakul and Chenjie Gu and Adam Roberts "DDSP: Differentiable Digital Signal Processing" ICLR 2020.

13.2.6 Deep clustering (Affinity) loss

`asteroid.losses.cluster.`**`deep_clustering_loss`**(*embedding*, *tgt_index*, *binary_mask=None*)

Compute the deep clustering loss defined in [1].

Parameters

- **`embedding`** (`torch.Tensor`) – Estimated embeddings. Expected shape $(batch, frequency * frame, embedding\_dim)$.
- **`tgt_index`** (`torch.Tensor`) – Dominating source index in each TF bin. Expected shape: $(batch, frequency, frame)$.
- **`binary_mask`** (`torch.Tensor`) – VAD in TF plane. Bool or Float. See asteroid.dsp.vad.ebased_vad.

Returns *torch.Tensor*. Deep clustering loss for every batch sample.

Examples

```
>>> import torch
>>> from asteroid.losses.cluster import deep_clustering_loss
>>> spk_cnt = 3
>>> embedding = torch.randn(10, 5*400, 20)
>>> targets = torch.LongTensor(10, 400, 5).random_(0, spk_cnt)
>>> loss = deep_clustering_loss(embedding, targets)
```

Reference [1] Zhong-Qiu Wang, Jonathan Le Roux, John R. Hershey "ALTERNATIVE OBJECTIVE FUNCTIONS FOR DEEP CLUSTERING"

Note: Be careful in viewing the embedding tensors. The target indices `tgt_index` are of shape $(batch, freq, frames)$. Even if the embedding is of shape $(batch, freq * frames, emb)$, the underlying view should be $(batch, freq, frames, emb)$ and not $(batch, frames, freq, emb)$.

13.3 Computing metrics

asteroid.metrics.**get_metrics**(*mix*, *clean*, *estimate*, *sample_rate=16000*, *metrics_list='all'*, *average=True*, *compute_permutation=False*, *ignore_metrics_errors=False*, *filename=None*)

Get speech separation/enhancement metrics from mix/clean/estimate.

Parameters

- **mix** (*np.array*) – mixture array.
- **clean** (*np.array*) – reference array.
- **estimate** (*np.array*) – estimate array.
- **sample_rate** (*int*) – sampling rate of the audio clips.
- **metrics_list** (*Union[List[str], str]*) – List of metrics to compute. Defaults to 'all' (['si_sdr', 'sdr', 'sir', 'sar', 'stoi', 'pesq']).
- **average** (*bool*) – Return dict([float]) if True, else dict([array]).
- **compute_permutation** (*bool*) – Whether to compute the permutation on estimate sources for the output metrics (default False)
- **ignore_metrics_errors** (*bool*) – Whether to ignore errors that occur in computing the metrics. A warning will be printed instead.
- **filename** (*str, optional*) – If computing a metric fails, print this filename along with the exception/warning message for debugging purposes.

Shape:

- mix: (D, N) or *(N,)*.
- clean: (K_source, N) or *(N,)*.
- estimate: (K_target, N) or *(N,)*.

Returns *dict* – Dictionary with all requested metrics, with *'input_'* prefix for metrics at the input (mixture against clean), no prefix at the output (estimate against clean). Output format depends on average.

Examples

```
>>> import numpy as np
>>> import pprint
>>> from asteroid.metrics import get_metrics
>>> mix = np.random.randn(1, 16000)
>>> clean = np.random.randn(2, 16000)
>>> est = np.random.randn(2, 16000)
>>> metrics_dict = get_metrics(mix, clean, est, sample_rate=8000,
...                            metrics_list='all')
```

(continues on next page)

(continued from previous page)

```
>>> pprint.pprint(metrics_dict)
{'input_pesq': 1.924380898475647,
 'input_sar': -11.67667585294225,
 'input_sdr': -14.88667106190552,
 'input_si_sdr': -52.43849784881705,
 'input_sir': -0.10419427290163795,
 'input_stoi': 0.015112115177091223,
 'pesq': 1.7713886499404907,
 'sar': -11.610963379923195,
 'sdr': -14.527246041125844,
 'si_sdr': -46.26557128489802,
 'sir': 0.4799929272243427,
 'stoi': 0.022023073540350643}
```

CHAPTER 14

Lightning Wrapper

As explained in *Training and Evaluation*, Asteroid provides a thin wrapper on the top of PyTorchLightning for training your models.

CHAPTER 15

Optimizers & Schedulers

15.1 Optimizers

Asteroid relies on torch_optimizer and `torch` for optimizers. We provide a simple `get` method that retrieves optimizers from string, which makes it easy to specify optimizers from the command line.

Here is a list of supported optimizers, retrievable from string:

- AccSGD
- AdaBound
- AdaMod
- DiffGrad
- Lamb
- NovoGrad
- PID
- QHAdam
- QHM
- RAdam
- SGDW
- Yogi
- Ranger
- RangerQH
- RangerVA
- Adam
- RMSprop

- SGD
- Adadelta
- Adagrad
- Adamax
- AdamW
- ASG

15.2 Schedulers

Asteroid provides step-wise learning schedulers, integrable to `pytorch-lightning` via `System`.

CHAPTER 16

DSP Modules

16.1 Beamforming

class asteroid.dsp.beamforming.**Beamformer**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

Base class for beamforming modules.

static apply_beamforming_vector(*bf_vector: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c86f50>*, *mix: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c86f90>*)

Apply the beamforming vector to the mixture. Output (batch, freqs, frames).

Parameters

- **bf_vector** – shape (batch, mics, freqs)
- **mix** – shape (batch, mics, freqs, frames).

static get_reference_mic_vects(*ref_mic*, *bf_mat: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b050>*, *target_scm: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b090> = None*, *noise_scm: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b0d0> = None*)

Return the reference channel indices over the batch.

Parameters

- **ref_mic** (*Optional[Union[int, torch.Tensor]]*) – The reference channel. If torch.Tensor (ndim>1), return it, it is the reference mic vector, If torch.LongTensor of size *batch*, select independent reference mic of the batch. If int, select the corresponding reference mic, If None, the optimal reference mics are computed with `get_optimal_reference_mic()`, If None, and either SCM is None, *ref_mic* is set to *0*,

- **bf_mat** – beamforming matrix of shape (batch, freq, mics, mics).
- **target_scm** (*torch.ComplexTensor*) – (batch, freqs, mics, mics).
- **noise_scm** (*torch.ComplexTensor*) – (batch, freqs, mics, mics).

Returns torch.LongTensor of size `batch` to select with the reference channel indices.

class asteroid.dsp.beamforming.**SDWMWFBeamformer**(*mu=1.0*)
Bases: *asteroid.dsp.beamforming.Beamformer*

forward(*mix: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b5d0>, target_scm: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b610>, noise_scm: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b650>, ref_mic: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b690>, <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b6d0>, int] = None*)
Compute and apply SDW-MWF beamformer.

$\mathbf{w} = (\Sigma_{ss} + \mu\Sigma_{nn})^{-1}\Sigma_{ss}$.

Parameters

- **mix** (*torch.ComplexTensor*) – shape (batch, mics, freqs, frames)
- **target_scm** (*torch.ComplexTensor*) – (batch, mics, mics, freqs)
- **noise_scm** (*torch.ComplexTensor*) – (batch, mics, mics, freqs)
- **ref_mic** (*int*) – reference microphone.

Returns Filtered mixture. torch.ComplexTensor (batch, freqs, frames)

class asteroid.dsp.beamforming.**GEVBeamformer**(**args*, ***kwargs*)
Bases: *asteroid.dsp.beamforming.Beamformer*

forward(*mix: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b790>, target_scm: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b7d0>, noise_scm: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b810>*)
Compute and apply the GEV beamformer.

$\mathbf{w} = MaxEig\{\Sigma_{nn}^{-1}\Sigma_{ss}\}$, where MaxEig extracts the eigenvector corresponding to the maximum eigenvalue (using the GEV decomposition).

Parameters

- **mix** – shape (batch, mics, freqs, frames)
- **target_scm** – (batch, mics, mics, freqs)
- **noise_scm** – (batch, mics, mics, freqs)

Returns Filtered mixture. torch.ComplexTensor (batch, freqs, frames)

class asteroid.dsp.beamforming.**RTFMVDRBeamformer**(**args*, ***kwargs*)
Bases: *asteroid.dsp.beamforming.Beamformer*

forward(*mix: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b1d0>, target_scm: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b210>, noise_scm: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b250>*)
Compute and apply MVDR beamformer from the speech and noise SCM matrices.

$\mathbf{w} = \frac{\Sigma_{nn}^{-1}\mathbf{a}}{\mathbf{a}^H\Sigma_{nn}^{-1}\mathbf{a}}$ where $\mathbf{a}$ is the ATF estimated from the target SCM.

Parameters

- **mix** (*torch.ComplexTensor*) – shape (batch, mics, freqs, frames)

- **target_scm** (*torch.ComplexTensor*) – (batch, mics, mics, freqs)
- **noise_scm** (*torch.ComplexTensor*) – (batch, mics, mics, freqs)

Returns Filtered mixture. torch.ComplexTensor (batch, freqs, frames)

from_rtf_vect(*mix: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b290>*, *rtf_vec: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b2d0>*, *noise_scm: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b310>*)

Compute and apply MVDR beamformer from the ATF vector and noise SCM matrix.

Parameters

- **mix** (*torch.ComplexTensor*) – shape (batch, mics, freqs, frames)
- **rtf_vec** (*torch.ComplexTensor*) – (batch, mics, freqs)
- **noise_scm** (*torch.ComplexTensor*) – (batch, mics, mics, freqs)

Returns Filtered mixture. torch.ComplexTensor (batch, freqs, frames)

class asteroid.dsp.beamforming.**SoudenMVDRBeamformer**(**args*, ***kwargs*)

Bases: *asteroid.dsp.beamforming.Beamformer*

forward(*mix: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b390>*, *target_scm: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b3d0>*, *noise_scm: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b410>*, *ref_mic: Union[<sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b450>, <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c8b490>, int] = 0*, *eps=1e-08*)

Compute and apply MVDR beamformer from the speech and noise SCM matrices. This class uses Souden's formulation [1].

$\mathbf{w} = \frac{\Sigma_{nn}^{-1}\Sigma_{ss}}{Tr\left(\Sigma_{nn}^{-1}\Sigma_{ss}\right)}\mathbf{u}$ where $\mathbf{a}$ is the steering vector.

Parameters

- **mix** (*torch.ComplexTensor*) – shape (batch, mics, freqs, frames)
- **target_scm** (*torch.ComplexTensor*) – (batch, mics, mics, freqs)
- **noise_scm** (*torch.ComplexTensor*) – (batch, mics, mics, freqs)
- **ref_mic** (*int*) – reference microphone.
- **eps** – numerical stabilizer.

Returns Filtered mixture. torch.ComplexTensor (batch, freqs, frames)

References [1] Souden, M., Benesty, J., & Affes, S. (2009). On optimal frequency-domain multichannel linear filtering for noise reduction. IEEE Transactions on audio, speech, and language processing, 18(2), 260-276.

class asteroid.dsp.beamforming.**SCM**(**args*, ***kwargs*)

Bases: sphinx.ext.autodoc.importer._MockObject

forward(*x: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c86dd0>*, *mask: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c86e10> = None*, *normalize: bool = True*)

See compute_scm().

16.2 LambdaOverlapAdd

class asteroid.dsp.**LambdaOverlapAdd**(*nnet*, *n_src*, *window_size*, *hop_size=None*, *window='hanning'*, *reorder_chunks=True*, *enable_grad=False*)

Bases: sphinx.ext.autodoc.importer._MockObject

Overlap-add with lambda transform on segments (not scriptable).

Segment input signal, apply lambda function (a neural network for example) and combine with OLA.

LambdaOverlapAdd can be used with `asteroid.separate` and the *asteroid-infer* CLI.

Parameters

- **nnet** (*callable*) – Function to apply to each segment.
- **n_src** (*Optional[int]*) – Number of sources in the output of nnet. If None, the number of sources is determined by the network's output, but some correctness checks cannot be performed.
- **window_size** (*int*) – Size of segmenting window.
- **hop_size** (*int*) – Segmentation hop size.
- **window** (*str*) – Name of the window (see scipy.signal.get_window) used for the synthesis.
- **reorder_chunks** – Whether to reorder each consecutive segment. This might be useful when *nnet* is permutation invariant, as source assignements might change output channel from one segment to the next (in classic speech separation for example). Reordering is performed based on the correlation between the overlapped part of consecutive segment.

ola_forward(*x*)

Heart of the class: segment signal, apply func, combine with OLA.

forward(*x*)

Forward module: segment signal, apply func, combine with OLA.

Parameters **x** (torch.Tensor) – waveform signal of shape (batch, 1, time).

Returns torch.Tensor – The output of the lambda OLA.

16.3 DualPath Processing

class asteroid.dsp.**DualPathProcessing**(*chunk_size*, *hop_size*)

Bases: sphinx.ext.autodoc.importer._MockObject

Perform Dual-Path processing via overlap-add as in DPRNN [1].

Parameters

- **chunk_size** (*int*) – Size of segmenting window.
- **hop_size** (*int*) – segmentation hop size.

References [1] Yi Luo, Zhuo Chen and Takuya Yoshioka. "Dual-path RNN: efficient long sequence modeling for time-domain single-channel speech separation" https://arxiv.org/abs/1910.06379

unfold(*x*)

Unfold the feature tensor from $(batch, channels, time)$ to $(batch, channels, chunksize, nchunks)$.

Parameters **x** (torch.Tensor) – feature tensor of shape $(batch, channels, time)$.

Returns torch.Tensor – spliced feature tensor of shape $(batch, channels, chunksize, nchunks)$.

fold(*x*, *output_size=None*)

Folds back the spliced feature tensor. Input shape $(batch, channels, chunksize, nchunks)$ to original shape $(batch, channels, time)$ using overlap-add.

Parameters

- **x** (torch.Tensor) – spliced feature tensor of shape $(batch, channels, chunksize, nchunks)$.
- **output_size** (*int, optional*) – sequence length of original feature tensor. If None, the original length cached by the previous call of *unfold()* will be used.

Returns torch.Tensor – feature tensor of shape $(batch, channels, time)$.

Note: *fold* caches the original length of the input.

---

static intra_process(*x*, *module*)

Performs intra-chunk processing.

Parameters

- **x** (torch.Tensor) – spliced feature tensor of shape (batch, channels, chunk_size, n_chunks).
- **module** (torch.nn.Module) – module one wish to apply to each chunk of the spliced feature tensor.

Returns torch.Tensor – processed spliced feature tensor of shape $(batch, channels, chunksize, nchunks)$.

Note: the module should have the channel first convention and accept a 3D tensor of shape $(batch, channels, time)$.

---

static inter_process(*x*, *module*)

Performs inter-chunk processing.

Parameters

- **x** (torch.Tensor) – spliced feature tensor of shape $(batch, channels, chunksize, nchunks)$.
- **module** (torch.nn.Module) – module one wish to apply between each chunk of the spliced feature tensor.

Returns x (torch.Tensor) – processed spliced feature tensor of shape $(batch, channels, chunksize, nchunks)$.

Note: the module should have the channel first convention and accept a 3D tensor of shape $(batch, channels, time)$.

16.4 Mixture Consistency

asteroid.dsp.**mixture_consistency**(*mixture: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a194efb50>, est_sources: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a194efb90>, src_weights: Optional[<sphinx.ext.autodoc.importer._MockObject object at 0x7f4a194efbd0>] = None, dim: int = 1*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a16da6650>

Applies mixture consistency to a tensor of estimated sources.

Parameters

- **mixture** (*torch.Tensor*) – Mixture waveform or TF representation.
- **est_sources** (*torch.Tensor*) – Estimated sources waveforms or TF representations.
- **src_weights** (*torch.Tensor*) – Consistency weight for each source. Shape needs to be broadcastable to *est_source*. We make sure that the weights sum up to 1 along dim *dim*. If *src_weights* is None, compute them based on relative power.
- **dim** (*int*) – Axis which contains the sources in *est_sources*.

Returns torch.Tensor with same shape as *est_sources*, after applying mixture consistency.

Examples

```
>>> # Works on waveforms
>>> mix = torch.randn(10, 16000)
>>> est_sources = torch.randn(10, 2, 16000)
>>> new_est_sources = mixture_consistency(mix, est_sources, dim=1)
>>> # Also works on spectrograms
>>> mix = torch.randn(10, 514, 400)
>>> est_sources = torch.randn(10, 2, 514, 400)
>>> new_est_sources = mixture_consistency(mix, est_sources, dim=1)
```

Note: This method can be used only in 'complete' separation tasks, otherwise the residual error will contain unwanted sources. For example, this won't work with the task *"sep_noisy"* from WHAM.

References Scott Wisdom et al. "Differentiable consistency constraints for improved deep speech enhancement", ICASSP 2019.

16.5 VAD

asteroid.dsp.vad.**ebased_vad**(*mag_spec*, *th_db: int = 40*)

Compute energy-based VAD from a magnitude spectrogram (or equivalent).

Parameters

- **mag_spec** (*torch.Tensor*) – the spectrogram to perform VAD on. Expected shape (batch, *, freq, time). The VAD mask will be computed independently for all the leading dimensions until the last two. Independent of the ordering of the last two dimensions.
- **th_db** (*int*) – The threshold in dB from which a TF-bin is considered silent.

Returns `torch.BoolTensor`, the VAD mask.

Examples

```
>>> import torch
>>> mag_spec = torch.abs(torch.randn(10, 2, 65, 16))
>>> batch_src_mask = ebased_vad(mag_spec)
```

16.6 Delta Features

asteroid.dsp.deltas.**compute_delta**(*feats: <sphinx.ext.autodoc.importer._MockObject object at 0x7f49f4cc9290>*, *dim: int = -1*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f49f4cc9310>

Compute delta coefficients of a tensor.

Parameters

- **feats** – Input features to compute deltas with.
- **dim** – feature dimension in the feats tensor.

Returns *Tensor* – Tensor of deltas.

Examples

```
>>> import torch
>>> phase = torch.randn(2, 257, 100)
>>> # Compute instantaneous frequency
>>> inst_freq = compute_delta(phase, dim=-1)
>>> # Or group delay
>>> group_delay = compute_delta(phase, dim=-2)
```

asteroid.dsp.deltas.**concat_deltas**(*feats: <sphinx.ext.autodoc.importer._MockObject object at 0x7f49f4cc9150>*, *order: int = 1*, *dim: int = -1*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f49f4cc92d0>

Concatenate delta coefficients of a tensor to itself.

Parameters

- **feats** – Input features to compute deltas with.
- **order** – Order of the delta e.g with order==2, compute delta of delta as well.
- **dim** – feature dimension in the feats tensor.

Returns *Tensor* – Concatenation of the features, the deltas and subsequent deltas.

Examples

```
>>> import torch
>>> phase = torch.randn(2, 257, 100)
>>> # Compute second order instantaneous frequency
>>> phase_and_inst_freq = concat_deltas(phase, order=2, dim=-1)
>>> # Or group delay
>>> phase_and_group_delay = concat_deltas(phase, order=2, dim=-2)
```

CHAPTER 17

Utils

17.1 Parser utils

Asteroid has its own argument parser (built on `argparse`) that handles dict-like structure, created from a config YAML file.

`asteroid.utils.parser_utils.`**`prepare_parser_from_dict`**(*dic*, *parser=None*)
Prepare an argparser from a dictionary.

Parameters

- **`dic`** (*`dict`*) – Two-level config dictionary with unique bottom-level keys.
- **`parser`** (*`argparse.ArgumentParser, optional`*) – If a parser already exists, add the keys from the dictionary on the top of it.

Returns *argparse.ArgumentParser* – Parser instance with groups corresponding to the first level keys and arguments corresponding to the second level keys with default values given by the values.

`asteroid.utils.parser_utils.`**`str_int_float`**(*value*)
Type to convert strings to int, float (in this order) if possible.

Parameters **`value`** (*`str`*) – Value to convert.

Returns *int, float, str* – Converted value.

`asteroid.utils.parser_utils.`**`str2bool`**(*value*)
Type to convert strings to Boolean (returns input if not boolean)

`asteroid.utils.parser_utils.`**`str2bool_arg`**(*value*)
Argparse type to convert strings to Boolean

`asteroid.utils.parser_utils.`**`isfloat`**(*value*)
Computes whether *value* can be cast to a float.

Parameters **`value`** (*`str`*) – Value to check.

Returns *bool* – Whether *value* can be cast to a float.

asteroid.utils.parser_utils.**isint**(*value*)
Computes whether *value* can be cast to an int

Parameters **value** (*str*) – Value to check.

Returns *bool* – Whether *value* can be cast to an int.

asteroid.utils.parser_utils.**parse_args_as_dict**(*parser*, *return_plain_args=False*, *args=None*)
Get a dict of dicts out of process *parser.parse_args()*

Top-level keys corresponding to groups and bottom-level keys corresponding to arguments. Under *'main_args'*, the arguments which don't belong to a argparse group (i.e main arguments defined before parsing from a dict) can be found.

Parameters

- **parser** (*argparse.ArgumentParser*) – ArgumentParser instance containing groups. Output of *prepare_parser_from_dict*.
- **return_plain_args** (*bool*) – Whether to return the output or *parser.parse_args()*.
- **args** (*list*) – List of arguments as read from the command line. Used for unit testing.

Returns *dict* – Dictionary of dictionaries containing the arguments. Optionally the direct output *parser.parse_args()*.

17.2 Torch utils

asteroid.utils.torch_utils.**to_cuda**(*tensors*)
Transfer tensor, dict or list of tensors to GPU.

Parameters **tensors** (torch.Tensor, list or dict) – May be a single, a list or a dictionary of tensors.

Returns torch.Tensor – Same as input but transferred to cuda. Goes through lists and dicts and transfers the torch.Tensor to cuda. Leaves the rest untouched.

asteroid.utils.torch_utils.**tensors_to_device**(*tensors*, *device*)
Transfer tensor, dict or list of tensors to device.

Parameters

- **tensors** (torch.Tensor) – May be a single, a list or a dictionary of tensors.
- **(** (*device*) – class: *torch.device*): the device where to place the tensors.

Returns Union [torch.Tensor, list, tuple, dict] – Same as input but transferred to device. Goes through lists and dicts and transfers the torch.Tensor to device. Leaves the rest untouched.

asteroid.utils.torch_utils.**get_device**(*tensor_or_module*, *default=None*)
Get the device of a tensor or a module.

Parameters

- **tensor_or_module** (*Union[torch.Tensor, torch.nn.Module]*) – The object to get the device from. Can be a `torch.Tensor`, a `torch.nn.Module`, or anything else that has a `device` attribute or a `parameters() -> Iterator[torch.Tensor]` method.
- **default** (*Optional[Union[str, torch.device]]*) – If the device can not be determined, return this device instead. If `None` (the default), raise a `TypeError` instead.

Returns *torch.device* – The device that `tensor_or_module` is on.

`asteroid.utils.torch_utils.`**`is_tracing`**`()`
Returns `True` in tracing (if a function is called during the tracing of code with `torch.jit.trace`) and `False` otherwise.

`asteroid.utils.torch_utils.`**`script_if_tracing`**`(`*`fn`*`)`
Compiles `fn` when it is first called during tracing. `torch.jit.script` has a non-negligible start up time when it is first called due to lazy-initializations of many compiler builtins. Therefore you should not use it in library code. However, you may want to have parts of your library work in tracing even if they use control flow. In these cases, you should use `@torch.jit.script_if_tracing` to substitute for `torch.jit.script`.

Parameters **`fn`** – A function to compile.

Returns If called during tracing, a `ScriptFunction` created by ‘ *torch.jit.script*‘ is returned. Otherwise, the original function `fn` is returned.

`asteroid.utils.torch_utils.`**`pad_x_to_y`**`(`*x: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c96150>, y: <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c96190>, axis: int = -1*`) → <sphinx.ext.autodoc.importer._MockObject object at 0x7f4a03c961d0>
Right-pad or right-trim first argument to have same size as second argument

Parameters

- **`x`** (*`torch.Tensor`*) – Tensor to be padded.
- **`y`** (*`torch.Tensor`*) – Tensor to pad *x* to.
- **`axis`** (*`int`*) – Axis to pad on.

Returns torch.Tensor, *x* padded to match *y*'s shape.

`asteroid.utils.torch_utils.`**`load_state_dict_in`**`(`*`state_dict`*, *`model`*`)`

Strictly loads state_dict in model, or the next submodel. Useful to load standalone model after training it with System.

Parameters

- **`state_dict`** (*`OrderedDict`*) – the state_dict to load.
- **`model`** (*`torch.nn.Module`*) – the model to load it into

Returns *torch.nn.Module* – model with loaded weights.

Note: Keys in a state_dict look like `object1.object2.layer_name.weight.etc` We first try to load the model in the classic way. If this fail we removes the first left part of the key to obtain `object2.layer_name.weight.etc`. Blindly loading with `strictly=False` should be done with some logging of the missing keys in the state_dict and the model.

`asteroid.utils.torch_utils.`**`are_models_equal`**`(`*`model1`*, *`model2`*`)`
Check for weights equality between models.

Parameters

- **`model1`** (*`nn.Module`*) – model instance to be compared.
- **`model2`** (*`nn.Module`*) – second model instance to be compared.

Returns *bool* – Whether all model weights are equal.

asteroid.utils.torch_utils.**jitable_shape**(*tensor*)
Gets shape of `tensor` as `torch.Tensor` type for jit compiler

Note: Returning `tensor.shape` of `tensor.size()` directly is not torchscript compatible as return type would not be supported.

Parameters **tensor** (*torch.Tensor*) – Tensor

Returns *torch.Tensor* – Shape of `tensor`

17.3 Hub utils

asteroid.utils.hub_utils.**cached_download**(*filename_or_url*)
Download from URL and cache the result in ASTEROID_CACHE.

Parameters **filename_or_url** (*str*) – Name of a model as named on the Zenodo Community page (ex: `"mpariente/ConvTasNet_WHAM!_sepclean"`), or model id from the Hugging Face model hub (ex: `"julien-c/DPRNNTasNet-ks16_WHAM_sepclean"`), or a URL to a model file (ex: `"https://zenodo.org/.../model.pth"`), or a filename that exists locally (ex: `"local/tmp_model.pth"`)

Returns str, normalized path to the downloaded (or not) model

asteroid.utils.hub_utils.**url_to_filename**(*url*)
Consistently convert `url` into a filename.

asteroid.utils.hub_utils.**model_list**
Get the public list of all the models on huggingface with an 'asteroid' tag.

17.4 Generic utils

asteroid.utils.generic_utils.**has_arg**(*fn*, *name*)
Checks if a callable accepts a given keyword argument.

Parameters

- **fn** (*callable*) – Callable to inspect.
- **name** (*str*) – Check if `fn` can be called with `name` as a keyword argument.

Returns *bool* – whether `fn` accepts a `name` keyword argument.

asteroid.utils.generic_utils.**flatten_dict**(*d*, *parent_key=''*, *sep='_'*)
Flattens a dictionary into a single-level dictionary while preserving parent keys. Taken from SO

Parameters

- **d** (*MutableMapping*) – Dictionary to be flattened.
- **parent_key** (*str*) – String to use as a prefix to all subsequent keys.
- **sep** (*str*) – String to use as a separator between two key levels.

Returns *dict* – Single-level dictionary, flattened.

asteroid.utils.generic_utils.**average_arrays_in_dic**(*dic*)
Take average of numpy arrays in a dictionary.

Parameters **dic** (*dict*) – Input dictionary to take average from

Returns *dict* – New dictionary with array averaged.

asteroid.utils.generic_utils.**get_wav_random_start_stop**(*signal_len*, *desired_len=32000*)
Get indexes for a chunk of signal of a given length.

Parameters

- **signal_len** (*int*) – length of the signal to trim.
- **desired_len** (*int*) – the length of [start:stop]

Returns *tuple* – random start integer, stop integer.

asteroid.utils.generic_utils.**unet_decoder_args**(*encoders*, *, *skip_connections*)
Get list of decoder arguments for upsampling (right) side of a symmetric u-net, given the arguments used to construct the encoder.

Parameters

- **encoders** (tuple of length *N* of tuples of (in_chan, out_chan, kernel_size, stride, padding)) – List of arguments used to construct the encoders
- **skip_connections** (*bool*) – Whether to include skip connections in the calculation of decoder input channels.

Returns tuple of length *N* of tuples of (in_chan, out_chan, kernel_size, stride, padding) – Arguments to be used to construct decoders

CHAPTER 18

Asteroid High-Level Contribution Guide

Asteroid is a Pytorch-based audio source separation toolkit that enables fast experimentation on common datasets.

18.1 The Asteroid Contribution Process

The Asteroid development process involves a healthy amount of open discussions between the core development team and the community.

Asteroid operates similar to most open source projects on GitHub. However, if you've never contributed to an open source project before, here is the basic process.

- **Figure out what you're going to work on.** The majority of open source contributions come from people scratching their own itches. However, if you don't know what you want to work on, or are just looking to get more acquainted with the project, here are some tips for how to find appropriate tasks:
  - Look through the issue tracker and see if there are any issues you know how to fix. Issues that are confirmed by other contributors tend to be better to investigate.
  - Join us on Slack and let us know you're interested in getting to know Asteroid. We're very happy to help out researchers and partners get up to speed with the codebase.
- **Figure out the scope of your change and reach out for design comments on a GitHub issue if it's large.** The majority of pull requests are small; in that case, no need to let us know about what you want to do, just get cracking. But if the change is going to be large, it's usually a good idea to get some design comments about it first.
  - If you don't know how big a change is going to be, we can help you figure it out! Just post about it on issues or Slack.
  - Some feature additions are very standardized; for example, lots of people add new datasets or architectures to Asteroid. Design discussion in these cases boils down mostly to, "Do we want this dataset/architecture?" Giving evidence for its utility, e.g., usage in peer reviewed papers, or existence in other frameworks, helps a bit when making this case.

  - Core changes and refactors can be quite difficult to coordinate, as the pace of development on Asteroid master is quite fast. Definitely reach out about fundamental or cross-cutting changes; we can often give guidance about how to stage such changes into more easily reviewable pieces.

- **Code it out!**
  - See the technical guide and read the code for advice for working with Asteroid in a technical form.
- **Open a pull request.**
  - If you are not ready for the pull request to be reviewed, tag it with [WIP]. We will ignore it when doing review passes. If you are working on a complex change, it's good to start things off as WIP, because you will need to spend time looking at CI results to see if things worked out or not.
  - Find an appropriate reviewer for your change. We have some folks who regularly go through the PR queue and try to review everything, but if you happen to know who the maintainer for a given subsystem affected by your patch is, feel free to include them directly on the pull request.
- **Iterate on the pull request until it's accepted!**
  - We'll try our best to minimize the number of review roundtrips and block PRs only when there are major issues. For the most common issues in pull requests, take a look at *Common Mistakes*.
  - Once a pull request is accepted and CI is passing, there is nothing else you need to do; we will merge the PR for you.

18.2 Getting Started

18.2.1 Proposing new features

New feature ideas are best discussed on a specific issue. Please include as much information as you can, any accompanying data, and your proposed solution. The Asteroid team and community frequently reviews new issues and comments where they think they can help. If you feel confident in your solution, go ahead and implement it.

18.2.2 Reporting Issues

If you've identified an issue, first search through the list of existing issues on the repo. If you are unable to find a similar issue, then create a new one. Supply as much information you can to reproduce the problematic behavior. Also, include any additional insights like the behavior you expect.

18.2.3 Implementing Features or Fixing Bugs

If you want to fix a specific issue, it's best to comment on the individual issue with your intent. However, we do not lock or assign issues except in cases where we have worked with the developer before. It's best to strike up a conversation on the issue and discuss your proposed solution. We can provide guidance that saves you time.

18.2.4 Adding Tutorials

Most our tutorials come from our team but we are very open to additional contributions. Have a notebook leveraging Asteroid? Open a PR to let us know!

18.2.5 Improving Documentation & Tutorials

We aim to produce high quality documentation and tutorials. On some occasions that content includes typos or bugs. If you find something you can fix, send us a pull request for consideration.

Take a look at the *Documentation* section to learn how our system works.

18.2.6 Participating in online discussions

You can find active discussions happening on our slack workspace.

18.2.7 Submitting pull requests to fix open issues

You can view a list of all open issues here. Commenting on an issue is a great way to get the attention of the team. From here you can share your ideas and how you plan to resolve the issue.

For more challenging issues, the team will provide feedback and direction for how to best solve the issue.

If you're not able to fix the issue itself, commenting and sharing whether you can reproduce the issue can be useful for helping the team identify problem areas.

18.2.8 Reviewing open pull requests

We appreciate your help reviewing and commenting on pull requests. Our team strives to keep the number of open pull requests at a manageable size, we respond quickly for more information if we need it, and we merge PRs that we think are useful. However, additional eyes on pull requests is always appreciated.

18.2.9 Improving code readability

Improve code readability helps everyone. We plan to integrate `black`/DeepSource in the CI process, but readability issues can still persist and we'll welcome your corrections.

18.2.10 Adding test cases to make the codebase more robust

Additional test coverage is **always** appreciated.

18.2.11 Promoting Asteroid

Your use of Asteroid in your projects, research papers, write ups, blogs, or general discussions around the internet helps to raise awareness for Asteroid and our growing community. Please reach out to us for support.

18.2.12 Triaging issues

If you feel that an issue could benefit from a particular tag or level of complexity comment on the issue and share your opinion. If an you feel an issue isn't categorized properly comment and let the team know.

18.3 About open source development

If this is your first time contributing to an open source project, some aspects of the development process may seem unusual to you.

- **There is no way to "claim" issues.** People often want to "claim" an issue when they decide to work on it, to ensure that there isn't wasted work when someone else ends up working on it. This doesn't really work too well in open source, since someone may decide to work on something, and end up not having time to do it. Feel free to give information in an advisory fashion, but at the end of the day, we will take running code and rough consensus.
- **There is a high bar for new functionality that is added.** Unlike in a corporate environment, where the person who wrote code implicitly "owns" it and can be expected to take care of it in the beginning of its lifetime, once a pull request is merged into an open source project, it immediately becomes the collective responsibility of all maintainers on the project. When we merge code, we are saying that we, the maintainers, are able to review subsequent changes and make a bugfix to the code. This naturally leads to a higher standard of contribution.

18.4 Common Mistakes To Avoid

- **Did you add tests?** (Or if the change is hard to test, did you describe how you tested your change?)
  - We have a few motivations for why we ask for tests:
    1. to help us tell if we break it later
    2. to help us tell if the patch is correct in the first place (yes, we did review it, but as Knuth says, "beware of the following code, for I have not run it, merely proven it correct")
  - When is it OK not to add a test? Sometimes a change can't be conveniently tested, or the change is so obviously correct (and unlikely to be broken) that it's OK not to test it. On the contrary, if a change is seems likely (or is known to be likely) to be accidentally broken, it's important to put in the time to work out a testing strategy.
- **Is your PR too long?** It's easier for us to review and merge small PRs. Difficulty of reviewing a PR scales nonlinearly with its size. You can try to split it up if possible, else it helps if there is a complete description of the contents of the PR: it's easier to review code if we know what's inside!
- **Comments for subtle things?** In cases where behavior of your code is nuanced, please include extra comments and documentation to allow us to better understand the intention of your code.
- **Did you add a hack?** Sometimes a hack is the right answer. But usually we will have to discuss it.
- **Do you want to touch a very core component?** In order to prevent major regressions, pull requests that touch core components receive extra scrutiny. Make sure you've discussed your changes with the team before undertaking major changes.
- **Want to add a new feature?** If you want to add new features, comment your intention on the related issue. Our team tries to comment on and provide feedback to the community. It's better to have an open discussion with the team and the rest of the community prior to building new features. This helps us stay aware of what you're working on and increases the chance that it'll be merged.
- **Did you touch unrelated code to the PR?** To aid in code review, please only include files in your pull request that are directly related to your changes.

18.5 Frequently asked questions

- **How can I contribute as a reviewer?** There is lots of value if community developer reproduce issues, try out new functionality, or otherwise help us identify or troubleshoot issues. Commenting on tasks or pull requests with your environment details is helpful and appreciated.
- **CI tests failed, what does it mean?** Maybe you need to merge with master or rebase with latest changes. Pushing your changes should re-trigger CI tests. If the tests persist, you'll want to trace through the error messages and resolve the related issues.

18.6 Attribution

This Contribution Guide is adapted from PyTorch's Contribution Guide available here.

CHAPTER 19

How to contribute

The general way to contribute to Asteroid is to fork the main repository on GitHub:

1. Fork the main repo and `git clone` it.
2. Make your changes, test them, commit them and push them to your fork.
3. You can open a pull request on GitHub when you're satisfied.

Things don't need to be perfect for PRs to be opened.

If you made changes to the source code, you'll want to try them out without installing asteroid everytime you change something. To do that, install asteroid in develop mode either with pip `pip install -e .[tests]` or with python `python setup.py develop`.

To avoid formatting roundtrips in PRs, Asteroid relies on *''black'* <https://github.com/psf/black>'_ and *''pre-commit-hooks'* <https://github.com/pre-commit/pre-commit-hooks>'_ to handle formatting for us. You'll need to install `requirements/dev.txt` and install git hooks with `pre-commit install`.

Here is a summary:

```
### Install
git clone your_fork_url
cd asteroid
pip install -r requirements/dev.txt
pip install -e .
pre-commit install  # To run black before commit

# Make your changes
# Test them locally
# Commit your changes
# Push your changes
# Open a PR!
```

19.1 Source code contributions

All contributions to the source code of asteroid should be documented and unit-tested. See here to run the tests with coverage reports. Docstrings follow the Google format, have a look at other docstrings in the codebase for examples. Examples in docstrings can be bery useful, don't hesitate to add some!

19.2 Writing new recipes.

Most new recipes should follow the standard format that is described here. We are not dogmatic about it, but another organization should be explained and motivated. We welcome any recipe on standard or new datasets, with standard or new architectures. You can even link a paper submission with a PR number if you'd like!

19.3 Improving the docs.

If you found a typo, think something could be more explicit etc... Improving the documentation is always welcome. The instructions to install dependencies and build the docs can be found here. Docstrings follow the Google format, have a look at other docstrings in the codebase for examples.

19.4 Coding style

We use pre-commit hooks to format the code using `black`. The code is checked for `black`- and `flake8`- compliance on every commit with GitHub actions. Remember, continuous integration is not here to be all green, be to help us see where to improve !

If you have any question, open an issue or join the slack, we'll be happy to help you.

CHAPTER 20

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

Symbols

A

G

H

I

J

K

L

M

N

O

P

Z