asteroid Documentation

Release 0.4.0alpha

Manuel Pariente et al.

Oct 07, 2020

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!

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/mpariente/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.

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. .. code-block:: BibTex

  @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}, eprint={2001.08662}, }

- The baseline paper, here. .. code-block:: BibTex

  @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.

4.4 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},
```

(continues on next page)

(continued from previous page)

```
   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 the Asteroid community in Zenodo. Have a look at the Zenodo page to choose which model you want to use.

Enjoy having pretrained models? **Please share your models** if you train some, we made it simple with the `asteroid-upload` CLI, 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')
```

Use the search page if you want to narrow your search.

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 do is to run

```
asteroid-upload exp/your_exp_dir/publish_dir --uploader "Name Here"
```

Ok, not really. First you need to register to Zenodo (Sign in with GitHub: ok), create a token and use it with the `--token` option of the CLI, or by setting the `ACCESS_TOKEN` environment variable. If you plan to upload more models (and you should :innocent:), you can fill in your infos in `uploader_info.yml` at the root, like this.

```
uploader: Manuel Pariente
affiliation: INRIA
git_username: mpariente
token: TOKEN_HERE
```

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 corresponding Zenodo page (ex: here).

CHAPTER 7

FAQ

7.1 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. Need to attach a DPRNN log here.

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.

7.2 How long does it take to train a model?

Need a log here.

7.3 Can I use the pretrained models for commercial purposes?

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

7.4 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

CHAPTER 8

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*.

8.1 LibriMix

8.2 Wsj0mix

8.3 WHAM!

8.4 WHAMR!

8.5 SMS-WSJ

8.6 KinectWSJMix

8.7 DNSDataset

8.8 MUSDB18

8.9 FUSS

8.10 AVSpeech

CHAPTER 9

Filterbank API

9.1 Filterbank, Encoder and Decoder

class asteroid.filterbanks.**Filterbank**(*n_filters*, *kernel_size*, *stride=None*)
Bases: sphinx.ext.autodoc.importer._MockObject

Base Filterbank class. Each subclass has to implement a *filters* property.

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.

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

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

filters
Abstract method for filters.

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.

forward(*waveform*)

Convolve input waveform with the filters from a filterbank. :param waveform: 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)
```

classmethod pinv_of(*filterbank*, ***kwargs*)

Returns an *Encoder*, pseudo inverse of a *Filterbank* or *Decoder*.

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.

Notes *padding* and *output_padding* arguments are directly passed to F.conv_transpose1d.

forward(*spec*)

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()*).

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

classmethod pinv_of(*filterbank*)

Returns an Decoder, pseudo inverse of a filterbank or Encoder.

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.
- **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

9.2 Learnable filterbanks

9.2.1 Free

class asteroid.filterbanks.free_fb.**FreeFB**(*n_filters*, *kernel_size*, *stride=None*, ***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.

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

References

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

filters
Abstract method for filters.

9.2.2 Analytic Free

class asteroid.filterbanks.analytic_free_fb.**AnalyticFreeFB**(*n_filters*, *kernel_size*, *stride=None*, ***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`.

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

References

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

filters
Abstract method for filters.

9.2.3 Parameterized Sinc

class asteroid.filterbanks.param_sinc_fb.**ParamSincFB**(*n_filters*, *kernel_size*, *stride=None*, *sample_rate=16000*, *min_low_hz=50*, *min_band_hz=50*)

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** (*int, 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". Submitted to ICASSP 2020. Manuel Pariente, Samuele Cornell, Antoine Deleforge, Emmanuel Vincent. https://arxiv.org/abs/1910.10400

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

filters
Compute filters from parameters

9.3 Fixed filterbanks

9.3.1 STFT

class asteroid.filterbanks.stft_fb.**STFTFB**(*n_filters*, *kernel_size*, *stride=None*, *window=None*, ***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())`.

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.

9.3.2 MP-GTFB

class asteroid.filterbanks.multiphase_gammatone_fb.**MultiphaseGammatoneFB**(*n_filters=128*, *kernel_size=16*, *sample_rate=8000*, *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: <*https://github.com/sp-uhh/mp-gtf*>

Parameters

- **n_filters** (*int*) – Number of filters.
- **kernel_size** (*int*) – Length of the filters.
- **sample_rate** (*int, 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.

asteroid.filterbanks.multiphase_gammatone_fb.**erb_scale_2_freq_hz**(*freq_erb*)
Convert frequency on ERB scale to frequency in Hertz

asteroid.filterbanks.multiphase_gammatone_fb.**freq_hz_2_erb_scale**(*freq_hz*)
Convert frequency in Hertz to frequency on ERB scale

asteroid.filterbanks.multiphase_gammatone_fb.**gammatone_impulse_response**(*samplerate_hz*, *len_sec*, *center_freq_hz*, *phase_shift*)
Generate single parametrized gammatone filter

asteroid.filterbanks.multiphase_gammatone_fb.**normalize_filters**(*filterbank*)
Normalizes a filterbank such that all filters have the same root mean square (RMS).

9.4 Transforms

9.4.1 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.take_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.take_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)

9.4.2 Complex transforms

asteroid.filterbanks.transforms.**angle**(*tensor*, *dim=-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.**apply_complex_mask**(*tf_rep*, *mask*, *dim=-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.`**`apply_mag_mask`**(*tf_rep*, *mask*, *dim=-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_real_mask`**(*tf_rep*, *mask*, *dim=-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.`**`check_complex`**(*tensor*, *dim=-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.`**`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.`**`ebased_vad`**(*mag_spec*, *th_db=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)
```

asteroid.filterbanks.transforms.**from_mag_and_phase**(*mag*, *phase*, *dim=-2*)
Return a complex-like torch tensor from magnitude and phase components.

Parameters

- **mag** (*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.**from_numpy**(*array*, *dim=-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.**from_torchaudio**(*tensor*, *dim=-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.**is_asteroid_complex**(*tensor*, *dim=-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.**is_torchaudio_complex**(*x*)
Check if tensor is Torchaudio-style complex-like (last dimension is 2).

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

Returns True if last dimension is 2, else False.

asteroid.filterbanks.transforms.**mul_c**(*inp*, *other*, *dim=-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.**take_mag**(*x*, *dim=-2*)
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.**to_numpy**(*tensor*, *dim=-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.**to_torchaudio**(*tensor*, *dim=-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.

CHAPTER 10

DNN building blocks

10.1 Convolutional blocks

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

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

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.**DCUMaskNet**(*encoders*, *decoders*, *mask_bound='tanh'*, ***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".

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*, *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 asteroid.masknn.norms for valid values.
- **activation** (*str, optional*) – Type of activation to use. See asteroid.masknn.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.**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 `asteroid.masknn.norms` for valid values.
- **activation** (*str, optional*) – Type of activation to use. See `asteroid.masknn.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.**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.**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'*, *kernel_size=None*)

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.

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*)

Parameters **mixture_w** (torch.Tensor) – Tensor of shape [batch, n_filters, n_frames]

Returns torch.Tensor – estimated mask of shape [batch, n_src, n_filters, n_frames]

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

Notes

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*)

Parameters **mixture_w** (torch.Tensor) – Tensor of shape [batch, n_filters, n_frames]

Returns torch.Tensor – estimated mask of shape [batch, n_src, n_filters, n_frames]

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

10.2 Recurrent blocks

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".

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()". *n_freqs - 1* must be 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

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

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, n_chans, n_freqs, time], *in_size* must be *n_chans * n_freqs*.
- **hid_size** (*int, optional*) – Number of units in RNN.

- **rnn_type** (*str, optional*) – Type of RNN to use. See `SingleRNN` for valid values.
- **norm_type** (*Optional[str], optional*) – Norm to use after linear. See `asteroid.masknn.norms` for valid values. (Not used in [1]).

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 0x7fbe91ec8978>*)
Input shape: [batch, . . . , time]

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'*, *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*)

Parameters **mixture_w** (torch.Tensor) – Tensor of shape [batch, n_filters, n_frames]

Returns

torch.Tensor estimated mask of shape [batch, n_src, n_filters, n_frames]

class asteroid.masknn.recurrent.**DPRNNBlock**(*in_chan*, *hid_size*, *norm_type='gLN'*, *bidirectional=True*, *rnn_type='LSTM'*, *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.**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.**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.**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.**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]

10.3 Norms

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

Wrapper class for pytorch BatchNorm1D and BatchNorm2D

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

Channel-wise Layer Normalization (chanLN).

forward(*x*)
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*)

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

Global Layer Normalization (globLN).

forward(*x*)
Applies forward pass.

Works for any input size > 2D.

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

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

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

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

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

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

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

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*.

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

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

CHAPTER 11

Models

11.1 Base classes

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).

get_model_args()

Arguments needed to re-instantiate the model.

postprocess_decoded(*decoded*)

Hook to perform transformations on the decoded, time domain representation (output of the decoder) before original shape reconstruction.

Parameters **decoded** (*Tensor of shape (batch, n_src, time)*) – Output of the decoder, before original shape reconstruction.

Returns Transformed *decoded*

postprocess_encoded(*tf_rep*)
Hook to perform transformations on the encoded, time-frequency domain representation (output of the encoder) before encoder activation is applied.

Parameters **tf_rep** (*Tensor of shape (batch, freq, time)*) – Output of the encoder, before encoder activation is applied.

Returns Transformed *tf_rep*

postprocess_masked(*masked_tf_rep*)
Hook to perform transformations on the masked time-frequency domain representation (result of masking in the time-frequency domain) before decoding.

Parameters **masked_tf_rep** (*Tensor of shape (batch, n_src, freq, time)*) – Masked time-frequency representation, before decoding.

Returns Transformed *masked_tf_rep*

postprocess_masks(*masks*)
Hook to perform transformations on the masks (output of the masker) before masks are applied.

Parameters **masks** (*Tensor of shape (batch, n_src, freq, time)*) – Output of the masker

Returns Transformed *masks*

class asteroid.models.base_models.**BaseModel**
Bases: sphinx.ext.autodoc.importer._MockObject

file_separate(*filename: str*, *output_dir=None*, *force_overwrite=False*, ***kwargs*) → None
Filename interface to *separate*.

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*.

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

numpy_separate(*wav: <sphinx.ext.autodoc.importer._MockObject object at 0x7fbe91e8a7b8>*, ***kwargs*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7fbe91e8a7f0>
Numpy interface to *separate*.

separate(*wav*, *output_dir=None*, *force_overwrite=False*, ***kwargs*)
Infer separated sources from input waveforms. Also supports filenames.

Parameters

- **wav** (*Union[torch.Tensor, numpy.ndarray, str]*) – waveform array/tensor. Shape: 1D, 2D or 3D tensor, time last.

- **output_dir** (*str*) – path to save all the wav files. If None, estimated sources will be saved next to the original ones.
- **force_overwrite** (*bool*) – whether to overwrite existing files.
- ****kwargs** – keyword arguments to be passed to *_separate*.

Returns

Union[torch.Tensor, numpy.ndarray, None], the estimated sources. (batch, n_src, time) or (n_src, time) w/o batch dim.

Note: By default, *separate* calls *_separate* which calls *forward*. For models whose *forward* doesn't return waveform tensors, overwrite *_separate* to return waveform tensors.

---

serialize()

Serialize model and output dictionary.

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

torch_separate(*wav:* <*sphinx.ext.autodoc.importer._MockObject object at 0x7fbe91e83c88*>, ***kwargs*) → <sphinx.ext.autodoc.importer._MockObject object at 0x7fbe91e83cc0>

Core logic of *separate*.

asteroid.models.base_models.**BaseTasNet**

alias of *asteroid.models.base_models.BaseEncoderMaskerDecoder*

11.2 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*, *fb_name='free'*, *kernel_size=16*, *n_filters=512*, *stride=8*, *encoder_activation=None*, ***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.
- **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] : "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.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*, ***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`.
- ****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

11.3 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.

Methods (all methods return the requests response): create_new_deposition change_metadata_in_deposition, upload_new_file_to_deposition publish_deposition get_deposition remove_deposition remove_all_depositions

---

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.

---

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'}]

}

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

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

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

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']*

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

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

upload_new_file_to_deposition(*dep_id*, *file*, *name=None*)
Upload one file to existing deposition. :param dep_id: deposition id. You cna get it with

r = create_new_deposition(); dep_id = r.json()['id']

Parameters

- **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)

asteroid.models.publisher.**display_one_level_dict**(*dic*)
Single level dict to HTML :param dic: :type dic: dict

Returns str for HTML-encoded single level dic

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.**make_metadata_from_model**(*model*)

Create Zenodo deposit metadata for a given publishable model. :param model: Dictionary with all infos needed to publish.

More info to come.

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

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.**two_level_dict_html**(*dic*)

Two-level dict to HTML. :param dic: two-level dict :type dic: dict

Returns str for HTML-encoded two level dic

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.**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)

CHAPTER 12

Losses & Metrics

class asteroid.losses.**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 (targets, est_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.

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)
```

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 (targets, est_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

tuple – `torch.Tensor`: The loss corresponding to the best permutation of size (batch,).

`torch.LongTensor`: The indexes of the best permutations.

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

Parameters

- **pair_wise_losses** (`torch.Tensor`) – Tensor of shape [batch, n_src, n_src]. Pairwise losses.
- **n_src** (*int*) – Number of sources.
- **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

tuple – `torch.Tensor`: The loss corresponding to the best permutation of size (batch,).

`torch.LongTensor`: The indexes of the best permutations.

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

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. torch.Tensor(loss_value)
- **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 (targets, est_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 reorder_source(*source*, *n_src*, *min_loss_idx*)
Reorder sources according to the best permutation.

Parameters

- **source** (*torch.Tensor*) – Tensor of shape [batch, n_src, time]
- **n_src** (*int*) – Number of sources.
- **min_loss_idx** (*torch.LongTensor*) – Tensor of shape [batch], each item is in [0, n_src!).

Returns torch.Tensor – Reordered sources of shape [batch, n_src, time].

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

class asteroid.losses.**SingleSrcPMSQE**(*window_name='sqrt_hann'*, *window_weight=1.0*, *bark_eq=True*, *gain_eq=True*, *sample_rate=16000*)

Bases: sphinx.ext.autodoc.importer._MockObject

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 Copyright 2019: University of Granada, Signal Processing, Multimedia Transmission and Speech/Audio Technologies (SigMAT) Group.

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.take_mag(stft(ref))
>>> est_spec = transforms.take_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.take_mag(stft(ref))
>>> est_spec = transforms.take_mag(stft(est))
>>> loss_value = loss_func(ref_spec, est_spec)
```

bark_freq_equalization(*ref_bark_spectra*, *deg_bark_spectra*)

This version is applied in the degraded directly.

forward(*est_targets*, *targets*, *pad_mask=None*)

Args

est_targets (torch.Tensor): Dimensions (B, T, F). Padded degraded power spectrum in time-frequency domain.

targets (torch.Tensor): Dimensions (B, T, F). Zero-Padded reference power spectrum in time-frequency domain.

pad_mask (torch.Tensor, optional): Dimensions (B, T, 1). Mask to indicate the padding frames. Defaults to all ones.

Dimensions B: Number of sequences in the batch. T: Number of time frames. F: Number of frequency bins.

Returns torch.tensor of shape (B,), wD + 0.309 * wDA

Notes Dimensions (B, F, T) are also supported by SingleSrcPMSQE but are less efficient because input tensors are transposed (not inplace).

Examples

static get_correction_factor(*window_name*)
Returns the power correction factor depending on the window.

asteroid.losses.**SingleSrcNegSTOI**
alias of *asteroid.losses.stoi.NegSTOILoss*

class asteroid.losses.**SingleSrcMultiScaleSpectral**(*n_filters=None*, *windows_size=None*, *hops_size=None*, *alpha=1.0*)
Bases: sphinx.ext.autodoc.importer._MockObject

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 (torch.Tensor): Expected shape [batch, time]. Batch of target estimates.

targets (torch.Tensor): Expected shape [batch, time]. Batch of training targets.

alpha (float) : Weighting factor for the log term

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 International Conference on Learning Representations ICLR 2020 $

class asteroid.losses.**PairwiseNegSDR**(*sdr_type*, *zero_mean=True*, *take_log=True*)

Bases: sphinx.ext.autodoc.importer._MockObject

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 (torch.Tensor): Expected shape [batch, n_src, time]. Batch of target estimates.

targets (torch.Tensor): Expected shape [batch, n_src, time]. Batch of training targets.

Returns torch.Tensor – with shape [batch, n_src, n_src]. 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.**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 x 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.filterbanks.transforms.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).

12.1 Permutation invariant training (PIT) made easy

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 (targets, est_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.

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)
```

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 (targets, est_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

tuple – `torch.Tensor`: The loss corresponding to the best permutation of size (batch,).

`torch.LongTensor`: The indexes of the best permutations.

static find_best_perm(*pair_wise_losses*, *n_src*, *perm_reduce=None*, ***kwargs*)

Find the best permutation, given the pair-wise losses.

Parameters

- **pair_wise_losses** (`torch.Tensor`) – Tensor of shape [batch, n_src, n_src]. Pairwise losses.
- **n_src** (*int*) – Number of sources.

- **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

tuple – `torch.Tensor`: The loss corresponding to the best permutation of size (batch,).

`torch.LongTensor`: The indexes of the best permutations.

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

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. torch.Tensor(loss_value)
- **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 (targets, est_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 reorder_source(*source*, *n_src*, *min_loss_idx*)

Reorder sources according to the best permutation.

Parameters

- **source** (*torch.Tensor*) – Tensor of shape [batch, n_src, time]
- **n_src** (*int*) – Number of sources.
- **min_loss_idx** (*torch.LongTensor*) – Tensor of shape [batch], each item is in [0, n_src!).

Returns torch.Tensor – Reordered sources of shape [batch, n_src, time].

12.2 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.

12.2.1 MSE

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

Measure pairwise mean square error on a batch.

Shape:

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

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 (torch.Tensor): Expected shape [batch, *]. The batch of target estimates.

targets (torch.Tensor): Expected shape [batch, *]. The batch of training targets.

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 (torch.Tensor): Expected shape [batch, *]. The batch of target estimates.

targets (torch.Tensor): Expected shape [batch, *]. The batch of training targets.

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)
```

12.2.2 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 (torch.Tensor): Expected shape [batch, n_src, time]. Batch of target estimates.

targets (torch.Tensor): Expected shape [batch, n_src, time]. Batch of training targets.

Returns torch.Tensor – with shape [batch, n_src, n_src]. 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** (*string*) – 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:
- **| 'mean'. 'none'** (*'none'*) – no reduction will be applied,
- **'mean'** – the sum of the output will be divided by the number of
- **in the output.** (*elements*) –

Shape:

est_targets (torch.Tensor): Expected shape [batch, time]. Batch of target estimates.

targets (torch.Tensor): Expected shape [batch, time]. Batch of training targets.

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** (*string*) – 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 (torch.Tensor): Expected shape [batch, time]. Batch of target estimates.

targets (torch.Tensor): Expected shape [batch, time]. Batch of training targets.

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.

12.2.3 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 Copyright 2019: University of Granada, Signal Processing, Multimedia Transmission and Speech/Audio Technologies (SigMAT) Group.

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.take_mag(stft(ref))
>>> est_spec = transforms.take_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.take_mag(stft(ref))
>>> est_spec = transforms.take_mag(stft(est))
>>> loss_value = loss_func(ref_spec, est_spec)
```

12.2.4 STOI

asteroid.losses.stoi.**NegSTOILoss**(**args*, ***kwargs*)

Negated Short Term Objective Intelligibility (STOI) metric, to be used as a loss function. Inspired from [1, 2, 3] but not exactly the same : cannot be used as the STOI metric directly (use pystoi instead). See Notes.

Parameters

- **sample_rate** (*int*) – sample rate of the audio files
- **use_vad** (*bool*) – Whether to use simple VAD (see Notes)
- **extended** (*bool*) – Whether to compute extended version [3].

Shapes: (time,) –> (1,) (batch, time) –> (batch,) (batch, n_src, time) –> (batch, n_src)

Returns torch.Tensor of shape (batch, *,), only the time dimension has been reduced.

Note: In the NumPy version, some kind of simple VAD was used to remove the silent frames before chunking the signal into short-term envelope vectors. We don't do the same here because removing frames in a batch is cumbersome and inefficient. If *use_vad* is set to True, instead we detect the silent frames and keep a mask tensor. At the end, the normalized correlation of short-term envelope vectors is masked using this mask (unfolded) and the mean is computed taking the mask values into account.

Examples

```
>>> import torch
>>> from asteroid.losses import PITLossWrapper
>>> targets = torch.randn(10, 2, 32000)
>>> est_targets = torch.randn(10, 2, 32000)
>>> loss_func = PITLossWrapper(NegSTOILoss(sample_rate=8000), pit_from='pw_pt')
>>> loss = loss_func(est_targets, targets)
```

References

[1] C.H.Taal, R.C.Hendriks, R.Heusdens, J.Jensen 'A Short-Time Objective Intelligibility Measure for Time-Frequency Weighted Noisy Speech', ICASSP 2010, Texas, Dallas.

[2] C.H.Taal, R.C.Hendriks, R.Heusdens, J.Jensen 'An Algorithm for Intelligibility Prediction of Time-Frequency Weighted Noisy Speech', IEEE Transactions on Audio, Speech, and Language Processing, 2011.

[3] Jesper Jensen and Cees H. Taal, 'An Algorithm for Predicting the Intelligibility of Speech Masked by Modulated Noise Maskers', IEEE Transactions on Audio, Speech and Language Processing, 2016.

12.2.5 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 (torch.Tensor): Expected shape [batch, time]. Batch of target estimates.

targets (torch.Tensor): Expected shape [batch, time]. Batch of training targets.

alpha (float) : Weighting factor for the log term

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 International Conference on Learning Representations ICLR 2020 $

12.2.6 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 x 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.filterbanks.transforms.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).

12.3 Computing metrics

asteroid.metrics.**get_metrics**(*mix*, *clean*, *estimate*, *sample_rate=16000*, *metrics_list='all'*, *average=True*, *compute_permutation=False*)

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

Parameters

- **mix** (*np.array*) – 'Shape(D, N)' or 'Shape(N,)'.
- **clean** (*np.array*) – 'Shape(K_source, N)' or 'Shape(N,)'.
- **estimate** (*np.array*) – 'Shape(K_target, N)' or 'Shape(N,)'.
- **sample_rate** (*int*) – sampling rate of the audio clips.
- **metrics_list** (*Union [str, list]*) – 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)

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')
>>> 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,
```

(continues on next page)

(continued from previous page)

```
 'si_sdr': -46.26557128489802,
 'sir': 0.4799929272243427,
 'stoi': 0.022023073540350643}
```

CHAPTER 13

Lightning Wrapper

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

CHAPTER 14

Optimizers & Schedulers

14.1 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

14.2 Schedulers

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

CHAPTER 15

DSP Modules

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.

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

Parameters

- **nnet** (*callable*) – Function to apply to each segment.
- **n_src** (*int*) – Number of sources in the output of nnet.
- **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.

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.

ola_forward(*x*)

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

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].

Args: chunk_size (int): Size of segmenting window. hop_size (int): segmentation hop size.

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

fold(*x*, *output_size=None*)

Folds back the spliced feature tensor.

Input shape (batch, channels, chunk_size, n_chunks) to original shape (batch, channels, time) using overlap-add.

Parameters

- **x** – (torch.Tensor): spliced feature tensor of shape (batch, channels, chunk_size, n_chunks).
- **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 *x* – (torch.Tensor): feature tensor of shape (batch, channels, time).

Note: *fold* caches the original length of the pr

---

static inter_process(*x*, *module*)

Performs inter-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 between each chunk of the spliced feature tensor.

Returns

x (torch.Tensor) –

processed spliced feature tensor of shape (batch, channels, chunk_size, n_chunks).

Note: the module should have the channel first convention and accept a 3D tensor of shape (batch, channels, time).

---

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

x (torch.Tensor) –

processed spliced feature tensor of shape (batch, channels, chunk_size, n_chunks).

Note: the module should have the channel first convention and accept a 3D tensor of shape (batch, channels, time).

---

unfold(*x*)

Unfold the feature tensor from

(batch, channels, time) to (batch, channels, chunk_size, n_chunks).

Parameters **x** – (torch.Tensor): feature tensor of shape (batch, channels, time).

Returns

x –

(torch.Tensor): spliced feature tensor of shape (batch, channels, chunk_size, n_chunks).

asteroid.dsp.**mixture_consistency**(*mixture*, *est_sources*, *src_weights=None*, *dim=1*)

Applies mixture consistency to a tensor of estimated sources.

Args 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.

Notes 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.

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)
```

References Scott Wisdom, John R Hershey, Kevin Wilson, Jeremy Thorpe, Michael Chinen, Brian Patton, and Rif A Saurous. "Differentiable consistency constraints for improved deep speech enhancement", ICASSP 2019.

CHAPTER 16

Utils

16.1 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.`**`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()*.

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.**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.**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.

16.2 Torch utils

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.**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.**pad_x_to_y**(*x*, *y*, *axis=-1*)
Pad 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.**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.**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.

16.3 Hub utils

asteroid.utils.hub_utils.**cached_download**(*filename_or_url*)
Download from URL with torch.hub 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 an 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.

16.4 Generic utils

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.**flatten_dict**(*d*, *parent_key=''*, *sep='_'*)
Flattens a dictionary into a single-level dictionary while preserving parent keys. Taken from https://stackoverflow.com/questions/6027558/ flatten-nested-dictionaries-compressing-keys?answertab=votes#tab-top

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.**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.**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.**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** (list 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 list of length *N* of tuples of (in_chan, out_chan, kernel_size, stride, padding) – Arguments to be used to construct decoders

CHAPTER 17

CLI

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.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.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

E

F

R

S

T

U

Z