nara_wpe Documentation

Release 0.0.0

Lukas Drude

Jan 17, 2020

Contents

CHAPTER 1

Weighted Prediction Error for speech dereverberation

Background noise and signal reverberation due to reflections in an enclosure are the two main impairments in acoustic signal processing and far-field speech recognition. This work addresses signal dereverberation techniques based on WPE for speech recognition and other far-field applications. WPE is a compelling algorithm to blindly dereverberate acoustic signals based on long-term linear prediction.

The main algorithm is based on the following paper: Yoshioka, Takuya, and Tomohiro Nakatani. “Generalization of multi-channel linear prediction methods for blind MIMO impulse response shortening.” IEEE Transactions on Audio, Speech, and Language Processing 20.10 (2012): 2707-2720.

CHAPTER 2

Content

- Iterative offline WPE/ block-online WPE/ recursive frame-online WPE
- All algorithms implemented both in Numpy and in TensorFlow (works with version *1.12.0*).
- Continuously tested with Python 2.7, 3.5 and 3.6.
- Automatically built documentation: nara-wpe.readthedocs.io
- Modular design to facilitate changes for further research

CHAPTER 3

Installation

Install it directly with Pip, if you just want to use it:

```
pip install nara_wpe
```

If you want to make changes or want the most recent version: Clone the repository and install it as follows:

```
git clone https://github.com/fgnt/nara_wpe.git
cd nara_wpe
pip install --editable .
```

Check the example notebook for further details. If you download the example notebook, you can listen to the input audio examples and to the dereverberated output too.

CHAPTER 4

Citation

To cite this implementation, you can cite the following paper:

```
@InProceedings{Drude2018NaraWPE,
  Title     = {{NARA-WPE}: A Python package for weighted prediction error␣
↪dereverberation in {Numpy} and {Tensorflow} for online and offline processing},
  Author    = {Drude, Lukas and Heymann, Jahn and Boeddeker, Christoph and Haeb-
↪Umbach, Reinhold},
  Booktitle = {13. ITG Fachtagung Sprachkommunikation (ITG 2018)},
  Year      = {2018},
  Month     = {Oct},
}
```

To view the paper see IEEE Xplore (PDF) or for a preview see Paderborn University RIS (PDF).

CHAPTER 5

Comparision with the NTT WPE implementation

The fairly recent John Hopkins University paper (Manohar, Vimal: Acoustic Modeling for Overlapping Speech Recognition: JHU CHiME-5 Challenge System, ICASSP 2019) reporting on their CHiME 5 challenge results dedicate an entire table to the comparison of the Nara-WPE implementation and the NTT WPE implementation. Their result is, that the Nara-WPE implementation is as least as good as the NTT WPE implementation in all their reported conditions.

CHAPTER 6

Development history

Since 2017-09-05 a TensorFlow implementation has been added to *nara_wpe*. It has been tested with a few test cases against the Numpy implementation.

The first version of the Numpy implementation was written in June 2017 while Lukas Drude and Kateřina Žmolíková resided in Nara, Japan. The aim was to have a publicly available implementation of Takuya Yoshioka's 2012 paper.

CHAPTER 7

Welcome to nara_wpe's documentation!

Table of contents:

7.1 nara_wpe package

7.1.1 Submodules

nara_wpe.benchmark_online_wpe module

benchmark_online_wpe.**config_iterator**()

nara_wpe.gradient_overrides module

nara_wpe.test_utils module

class test_utils.**QuietTestRunner**
Bases: object

run(*suite*)

test_utils.**repeat_with_success_at_least**(*times*, *min_success*)
Decorator for multiple trial of the test case.

The decorated test case is launched multiple times. The case is judged as passed at least specified number of trials. If the number of successful trials exceeds *min_success*, the remaining trials are skipped.

Parameters

- **times** (*int*) – The number of trials.
- **min_success** (*int*) – Threshold that the decorated test case is regarded as passed.

test_utils.**retry**(*times*)
Decorator that imposes the test to be successful at least once.

Decorated test case is launched multiple times. The case is regarded as passed if it is successful at least once.

Note: In current implementation, this decorator grasps the failure information of each trial.

Parameters **times** (*int*) – The number of trials.

nara_wpe.tf_wpe module

tf_wpe.**batched_block_wpe_step**(*Y*, *inverse_power*, *num_frames*, *taps=10*, *delay=3*, *mode='inv'*, *block_length_in_seconds=2.0*, *forgetting_factor=0.7*, *fft_shift=256*, *sampling_rate=16000*)
Batched single WPE step. More suited for backpropagation.

Parameters

- **Y** (*tf.Tensor*) – Complex valued STFT signal with shape (B, F, D, T)
- **inverse_power** (*tf.Tensor*) – Power signal with shape (B, F, T)
- **num_frames** (*tf.Tensor*) – Number of frames for each signal in the batch
- **taps** (*int, optional*) – Filter order
- **delay** (*int, optional*) – Delay as a guard interval, such that X does not become zero.
- **mode** (*str, optional*) – Specifies how R^-1@r is calculate: "inv" calculates the inverse of R directly and then uses matmul "solve" solves Rx=r for x
- **block_length_in_seconds** (*float, optional*) – Length of each block in seconds
- **forgetting_factor** (*float, optional*) – Forgetting factor for the signal statistics between the blocks
- **fft_shift** (*int, optional*) – Shift used for the STFT.
- **sampling_rate** (*int, optional*) – Sampling rate of the observed signal.

Returns Dereverberated signal of shape B, (F, D, T)

tf_wpe.**batched_recursive_wpe**(*Y*, *power_estimate*, *alpha*, *num_frames*, *taps=10*, *delay=2*, *only_use_final_filters=False*)
Batched single WPE step. More suited for backpropagation.

Parameters

- **Y** (*tf.Tensor*) – Observed signal of shape (B, T, F, D)
- **power_estimate** (*tf.Tensor*) – Estimate for the clean signal PSD of shape (B, T, F)
- **alpha** (*float*) – Smoothing factor for the recursion
- **num_frames** (*tf.Tensor*) – Number of frames for each signal in the batch
- **K** (*int, optional*) – Number of filter taps.
- **delay** (*int, optional*) – Delay

- **only_use_final_filters** (*bool, optional*) – Applies only the final estimated filter coefficients to the whole signal. This is for debugging purposes only and makes this method a offline one.

Returns Dereverberated signal of shape (B, T, F, D)

tf_wpe.**batched_wpe**(*Y*, *num_frames*, *taps=10*, *delay=3*, *iterations=3*, *mode='inv'*)

Batched version of iterative WPE.

Parameters

- **Y** (*tf.Tensor*) – Observed signal with shape (B, F, D, T)
- **num_frames** (*tf.Tensor*) – Number of frames for each signal in the batch
- **taps** (*int, optional*) – Defaults to 10. Number of filter taps.
- **delay** (*int, optional*) – Defaults to 3.
- **iterations** (*int, optional*) – Defaults to 3.
- **mode** (*str, optional*) – Specifies how R^-1@r is calculate: "inv" calculates the inverse of R directly and then uses matmul "solve" solves Rx=r for x

Returns Dereverberated signal of shape (B, F, D, T).

Return type tf.Tensor

tf_wpe.**batched_wpe_step**(*Y*, *inverse_power*, *num_frames*, *taps=10*, *delay=3*, *mode='inv'*, *Y_stats=None*)

Batched single WPE step. More suited for backpropagation.

Parameters

- **Y** (*tf.Tensor*) – Complex valued STFT signal with shape (B, F, D, T)
- **inverse_power** (*tf.Tensor*) – Power signal with shape (B, F, T)
- **num_frames** (*tf.Tensor*) – Number of frames for each signal in the batch
- **taps** (*int, optional*) – Filter order
- **delay** (*int, optional*) – Delay as a guard interval, such that X does not become zero.
- **mode** (*str, optional*) – Specifies how R^-1@r is calculate: "inv" calculates the inverse of R directly and then uses matmul "solve" solves Rx=r for x
- **Y_stats** (*tf.Tensor or None, optional*) – Complex valued STFT signal with shape (F, D, T) use to calculate the signal statistics (i.e. correlation matrix/vector). If None, Y is used. Otherwise it's usually a segment of Y

Returns Dereverberated signal of shape B, (F, D, T)

tf_wpe.**block_wpe_step**(*Y*, *inverse_power*, *taps=10*, *delay=3*, *mode='inv'*, *block_length_in_seconds=2.0*, *forgetting_factor=0.7*, *fft_shift=256*, *sampling_rate=16000*)

Applies wpe in a block-wise fashion.

Parameters

- **Y** (*tf.Tensor*) – Complex valued STFT signal with shape (F, D, T)
- **inverse_power** (*tf.Tensor*) – Power signal with shape (F, T)
- **taps** (*int, optional*) – Defaults to 10.
- **delay** (*int, optional*) – Defaults to 3.

- **mode** (*str, optional*) – Specifies how R^-1@r is calculate: "inv" calculates the inverse of R directly and then uses matmul "solve" solves Rx=r for x
- **block_length_in_seconds** (*float, optional*) – Length of each block in seconds
- **forgetting_factor** (*float, optional*) – Forgetting factor for the signal statistics between the blocks
- **fft_shift** (*int, optional*) – Shift used for the STFT.
- **sampling_rate** (*int, optional*) – Sampling rate of the observed signal.

tf_wpe.**get_correlations**(*Y*, *inverse_power*, *taps*, *delay*)

Calculates weighted correlations of a window of length taps

Parameters

- **Y** (*tf.Ttensor*) – Complex-valued STFT signal with shape (F, D, T)
- **inverse_power** (*tf.Tensor*) – Weighting factor with shape (F, T)
- **taps** (*int*) – Lenghts of correlation window
- **delay** (*int*) – Delay for the weighting factor

Returns Correlation matrix of shape (F, taps*D, taps*D) tf.Tensor: Correlation vector of shape (F, taps*D)

Return type tf.Tensor

tf_wpe.**get_correlations_for_single_frequency**(*Y*, *inverse_power*, *taps*, *delay*)

Calculates weighted correlations of a window of length taps for one freq.

Parameters

- **Y** (*tf.Ttensor*) – Complex-valued STFT signal with shape (D, T)
- **inverse_power** (*tf.Tensor*) – Weighting factor with shape (T)
- **K** (*int*) – Lenghts of correlation window
- **delay** (*int*) – Delay for the weighting factor

Returns Correlation matrix of shape (taps*D, taps*D) tf.Tensor: Correlation vector of shape (D, taps*D)

Return type tf.Tensor

tf_wpe.**get_filter_matrix_conj**(*Y*, *correlation_matrix*, *correlation_vector*, *taps*, *delay*, *mode='solve'*)

Calculate (conjugate) filter matrix based on correlations for one freq.

Parameters

- **Y** (*tf.Tensor*) – Complex-valued STFT signal of shape (D, T)
- **correlation_matrix** (*tf.Tensor*) – Correlation matrix (taps*D, taps*D)
- **correlation_vector** (*tf.Tensor*) – Correlation vector (D, taps*D)
- **K** (*int*) – Number of filter taps
- **delay** (*int*) – Delay
- **mode** (*str, optional*) – Specifies how R^-1@r is calculate: "inv" calculates the inverse of R directly and then uses matmul "solve" solves Rx=r for x

Raises ValueError – Unknown mode specified

Returns (Conjugate) filter Matrix

Return type tf.Tensor

tf_wpe.**get_power**(*signal*, *axis=-2*)
Calculates power for *signal*

Parameters

- **signal** (*tf.Tensor*) – Single frequency signal with shape (D, T) or (F, D, T).
- **axis** – reduce_mean axis

Returns Power with shape (T,) or (F, T)

Return type tf.Tensor

tf_wpe.**get_power_inverse**(*signal*)
Calculates inverse power for *signal*

Parameters

- **signal** (*tf.Tensor*) – Single frequency signal with shape (D, T).
- **psd_context** – context for power estimation

Returns Inverse power with shape (T,)

Return type tf.Tensor

tf_wpe.**get_power_online**(*signal*)
Calculates power for *signal*

Parameters **signal** (*tf.Tensor*) – Signal with shape (F, D, T).

Returns Power with shape (F,)

Return type tf.Tensor

tf_wpe.**online_wpe_step**(*input_buffer*, *power_estimate*, *inv_cov*, *filter_taps*, *alpha*, *taps*, *delay*)
One step of online dereverberation

Parameters

- **input_buffer** (*tf.Tensor*) – Buffer of shape (taps+delay+1, F, D)
- **power_estimate** (*tf.Tensor*) – Estimate for the current PSD
- **inv_cov** (*tf.Tensor*) – Current estimate of R^-1
- **filter_taps** (*tf.Tensor*) – Current estimate of filter taps (F, taps*D, taps)
- **alpha** (*float*) – Smoothing factor
- **taps** (*int*) – Number of filter taps
- **delay** (*int*) – Delay in frames

Returns Dereverberated frame of shape (F, D) tf.Tensor: Updated estimate of R^-1 tf.Tensor: Updated estimate of the filter taps

Return type tf.Tensor

tf_wpe.**perform_filter_operation**(*Y*, *filter_matrix_conj*, *taps*, *delay*)

```
>>> D, T, taps, delay = 1, 10, 2, 1
>>> tf.enable_eager_execution()
>>> Y = tf.ones([D, T])
>>> filter_matrix_conj = tf.ones([taps, D, D])
>>> X = perform_filter_operation_v2(Y, filter_matrix_conj, taps, delay)
>>> X.shape
TensorShape([Dimension(1), Dimension(10)])
>>> X.numpy()
array([[ 1.,  0., -1., -1., -1., -1., -1., -1., -1., -1.]], dtype=float32)
```

tf_wpe.**recursive_wpe**(*Y*, *power_estimate*, *alpha*, *taps=10*, *delay=2*, *only_use_final_filters=False*)

Applies WPE in a framewise recursive fashion.

Parameters

- **Y** (*tf.Tensor*) – Observed signal of shape (T, F, D)
- **power_estimate** (*tf.Tensor*) – Estimate for the clean signal PSD of shape (T, F)
- **alpha** (*float*) – Smoothing factor for the recursion
- **taps** (*int, optional*) – Number of filter taps.
- **delay** (*int, optional*) – Delay
- **only_use_final_filters** (*bool, optional*) – Applies only the final estimated filter coefficients to the whole signal. This is for debugging purposes only and makes this method a offline one.

Returns Enhanced signal

Return type tf.Tensor

tf_wpe.**single_frequency_wpe**(*Y*, *taps=10*, *delay=3*, *iterations=3*, *mode='inv'*)

WPE for a single frequency.

Parameters

- **Y** – Complex valued STFT signal with shape (D, T)
- **taps** – Number of filter taps
- **delay** – Delay as a guard interval, such that X does not become zero.
- **iterations** –
- **mode** (*str, optional*) – Specifies how R^-1@r is calculate: "inv" calculates the inverse of R directly and then uses matmul "solve" solves Rx=r for x

Returns:

tf_wpe.**wpe**(*Y*, *taps=10*, *delay=3*, *iterations=3*, *mode='inv'*)

WPE for all frequencies at once. Use this for regular processing.

Parameters

- **Y** (*tf.Tensor*) – Observed signal with shape (F, D, T)
- **num_frames** (*tf.Tensor*) – Number of frames for each signal in the batch
- **taps** (*int, optional*) – Defaults to 10. Number of filter taps.
- **delay** (*int, optional*) – Defaults to 3.
- **iterations** (*int, optional*) – Defaults to 3.

- **mode** (*str, optional*) – Specifies how R^-1@r is calculated: "inv" calculates the inverse of R directly and then uses matmul "solve" solves Rx=r for x

Returns Dereverberated signal tf.Tensor: Latest estimation of the clean speech PSD

Return type tf.Tensor

tf_wpe.**wpe_step**(*Y*, *inverse_power*, *taps=10*, *delay=3*, *mode='inv'*, *Y_stats=None*)

Single step of 'wpe'. More suited for backpropagation.

Parameters

- **Y** (*tf.Tensor*) – Complex valued STFT signal with shape (F, D, T)
- **inverse_power** (*tf.Tensor*) – Power signal with shape (F, T)
- **taps** (*int, optional*) – Filter order
- **delay** (*int, optional*) – Delay as a guard interval, such that X does not become zero.
- **mode** (*str, optional*) – Specifies how R^-1@r is calculate: "inv" calculates the inverse of R directly and then uses matmul "solve" solves Rx=r for x
- **Y_stats** (*tf.Tensor or None, optional*) – Complex valued STFT signal with shape (F, D, T) use to calculate the signal statistics (i.e. correlation matrix/vector). If None, Y is used. Otherwise it's usually a segment of Y

Returns Dereverberated signal of shape (F, D, T)

nara_wpe.utils module

nara_wpe.wpe module

7.1.2 Module contents

Used by autodoc_mock_imports.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

n

t

Index