torch$_{specinv}$

Release 0.1

Jul 08, 2023

Package Reference

1 Installation 3

2 Getting Started 5

3 Indices and Tables 9

Python Module Index 11

Index 13

A major direction of Deep Learning in audio, especially generative models, is using features in frequency domain because directly model raw time signal is hard. But this require an extra process to convert the predicted spectrogram (magnitude-only in most situation) back to time domain.

To help researcher no need to care this post-precessing step, this package provide some useful and classic spectrogram inversion algorithms. These algorithms are selected base on their performance and high parallelizability, and can even be integrated in your model training process.

We hope this tool can serve as a standard, making fair comparison of different audio generation models.

CHAPTER 1

Installation

1.1 PyPi

First Install PyTorch with the desired cpu/gpu support and version >= 0.4.1. Then install via pip:

```
pip install torch_specinv
```

or:

```
pip install git+https://github.com/yoyololicon/spectrogram-inversion
```

to get the latest version.

CHAPTER 2

Getting Started

The following example estimated the time signal given only the magnitude information of an audio file.

```
import torch
import librosa
from torch_specinv import griffin_lim
from torch_specinv.metrics import spectral_convergence as SC

y, sr = librosa.load(librosa.util.example_audio_file())
y = torch.from_numpy(y)
windowsize = 2048
window = torch.hann_window(windowsize)
S = torch.stft(y, windowsize, window=window)

# discard phase information
mag = S.pow(2).sum(2).sqrt()

# move to gpu memory for faster computation
mag = mag.cuda()

yhat = griffin_lim(mag, maxiter=100, alpha=0.3, window=window)

# check convergence
mag_hat = torch.stft(yhat, windowsize, window=window).pow(2).sum(2).sqrt()
print(SC(mag_hat, mag))
```

Reconstruct from other spectral representation:

```
from librosa.filters import mel
from torch_specinv import L_BFGS

filter_banks = torch.from_numpy(mel(sr, windowsize)).cuda()

def trsfn(x):
   S = torch.stft(x, windowsize, window=window).pow(2).sum(2).sqrt()
```

(continues on next page)

(continued from previous page)

```
    mel_S = filter_banks @ S
    return torch.log1p(mel_S)

y = y.cuda()
mag = trsfn(y)
yhat = L_BFGS(mag, trsfn, len(y))
```

2.1 torch_specinv.methods

torch_specinv.methods.**ADMM**(*spec*, *max_iter=1000*, *tol=1e-06*, *rho=0.1*, *verbose=1*, *eva_iter=10*, *metric='sc'*, ***stft_kwargs*)

Reconstruct spectrogram phase using Griffin–Lim Like Phase Recovery via Alternating Direction Method of Multipliers .

Parameters

- **spec** (*Tensor*) – the input tensor of size $(N \times T)$ (magnitude) or $(N \times T \times 2)$ (complex input). If a magnitude spectrogram is given, the phase will first be intialized using *torch_specinv.methods.phase_init()*; otherwise start from the complex input.
- **max_iter** (*int*) – maximum number of iterations before timing out.
- **tol** (*float*) – tolerance of the stopping condition base on L2 loss. Default: `1e-6`
- **rho** (*float*) – non-negative speedup parameter. Small value is preferable when the input spectrogram is noisy (inperfect); set it to 1 will behave similar to `griffin_lim`. Default: `0.1`
- **verbose** (*bool*) – whether to be verbose. Default: `True`
- **eva_iter** (*int*) – steps size for evaluation. After each step, the function defined in `metric` will evaluate. Default: `10`
- **metric** (*str*) – evaluation function. Currently available functions: `'sc'` (spectral convergence), `'snr'` or `'ser'`. Default: `'sc'`
- ****stft_kwargs** – other arguments that pass to `torch.stft()`.

Returns A 1d tensor converted from the given spectrogram

torch_specinv.methods.**L_BFGS**(*spec*, *transform_fn*, *samples=None*, *init_x0=None*, *outer_max_iter=1000*, *tol=1e-06*, *verbose=1*, *eva_iter=10*, *metric='sc'*, ***kwargs*)

Reconstruct spectrogram phase using Inversion of Auditory Spectrograms, Traditional Spectrograms, and Other Envelope Representations, where I directly use the `torch.optim.LBFGS` optimizer provided in PyTorch. This method doesn't restrict to traditional short-time Fourier Transform, but any kinds of presentation (ex: Mel-scaled Spectrogram) as long as the transform function is differentiable.

Parameters

- **spec** (*Tensor*) – the input presentation.
- **transform_fn** – a function that has the form `spec = transform_fn(x)` where x is an 1d tensor.
- **samples** (*int, optional*) – number of samples in time domain. Default: `None`
- **init_x0** (*Tensor, optional*) – an 1d tensor that make use as initial time domain samples. If not provided, will use random value tensor with length equal to `samples`.

- **outer_max_iter** (*int*) – maximum number of iterations before timing out.
- **tol** (*float*) – tolerance of the stopping condition base on L2 loss. Default: `1e-6`.
- **verbose** (*bool*) – whether to be verbose. Default: `True`
- **eva_iter** (*int*) – steps size for evaluation. After each step, the function defined in `metric` will evaluate. Default: `10`
- **metric** (*str*) – evaluation function. Currently available functions: `'sc'` (spectral convergence), `'snr'` or `'ser'`. Default: `'sc'`
- ****kwargs** – other arguments that pass to `torch.optim.LBFGS`.

Returns A 1d tensor converted from the given presentation

torch_specinv.methods.**RTISI_LA**(*spec*, *look_ahead=-1*, *asymmetric_window=False*, *max_iter=25*, *alpha=0.99*, *verbose=1*, ***stft_kwargs*)

Reconstruct spectrogram phase using Real-Time Iterative Spectrogram Inversion with Look Ahead (RTISI-LA).

Parameters

- **spec** (*Tensor*) – the input tensor of size $(N \times T)$ (magnitude).
- **look_ahead** (*int*) – how many future frames will be consider. `-1` will set it to `(win_length - 1) / hop_length`, `0` will disable look-ahead strategy and fall back to original RTISI algorithm. Default: `-1`
- **asymmetric_window** (*bool*) – whether to apply asymmetric window on the first iteration for new coming frame.
- **max_iter** (*int*) – number of iterations for each step.
- **alpha** (*float*) – speedup parameter used in Fast Griffin-Lim, set it to zero will disable it. Default: `0`
- **verbose** (*bool*) – whether to be verbose. Default: `True`
- ****stft_kwargs** – other arguments that pass to `torch.stft()`.

Returns A 1d tensor converted from the given spectrogram

torch_specinv.methods.**griffin_lim**(*spec*, *max_iter=200*, *tol=1e-06*, *alpha=0.99*, *verbose=True*, *eva_iter=10*, *metric='sc'*, ***stft_kwargs*)

Reconstruct spectrogram phase using the will known Griffin-Lim algorithm and its variation, Fast Griffin-Lim.

Parameters

- **spec** (*Tensor*) – the input tensor of size $(N \times T)$ (magnitude) or $(N \times T \times 2)$ (complex input). If a magnitude spectrogram is given, the phase will first be intialized using *torch_specinv.methods.phase_init()*; otherwise start from the complex input.
- **max_iter** (*int*) – maximum number of iterations before timing out.
- **tol** (*float*) – tolerance of the stopping condition base on L2 loss. Default: `1e-6`
- **alpha** (*float*) – speedup parameter used in Fast Griffin-Lim, set it to zero will disable it. Default: `0`
- **verbose** (*bool*) – whether to be verbose. Default: `True`
- **eva_iter** (*int*) – steps size for evaluation. After each step, the function defined in *metric* will evaluate. Default: `10`
- **metric** (*str*) – evaluation function. Currently available functions: `'sc'` (spectral convergence), `'snr'` or `'ser'`. Default: `'sc'`

- ****stft_kwargs** – other arguments that pass to `torch.stft()`

Returns A 1d tensor converted from the given spectrogram

`torch_specinv.methods.`**`phase_init`**(*spec*, ***stft_kwargs*)

A phase initialize function that can be seen as a simplified version of Single Pass Spectrogram Inversion.

Parameters

- **spec** (*Tensor*) – the input tensor of size $(* \times N \times T)$ (magnitude).
- ****stft_kwargs** – other arguments that pass to `torch.stft()`

Returns The estimated complex value spectrogram of size $(N \times T \times 2)$

2.2 torch_specinv.metrics

`torch_specinv.metrics.`**`sc`**(*input*, *target*)

The Spectral Convergence score is calculated as follow:

$$\mathcal{C}(\hat{\mathbf{S}}, \mathbf{S}) = \frac{\|\hat{\mathbf{S}} - \mathbf{S}\|_{Fro}}{\|\mathbf{S}\|_{Fro}}$$

Returns scalar output in db scale.

`torch_specinv.metrics.`**`ser`**(*input*, *target*)

The Signal-to-Error Ratio (SER) is calculated as follow:

$$SER(\hat{\mathbf{S}}, \mathbf{S}) = 10 \log_{10} \frac{\sum \hat{s}_i^2}{\sum (\hat{s}_i - s_i)^2}$$

Returns scalar output.

`torch_specinv.metrics.`**`snr`**(*input*, *target*)

The Signal-to-Noise Ratio (SNR) is calculated as follow:

$$SNR(\hat{\mathbf{S}}, \mathbf{S}) = 10 \log_{10} \frac{1}{\sum (\frac{\hat{s}_i}{\|\hat{\mathbf{S}}\|_{Fro}} - \frac{s_i}{\|\mathbf{S}\|_{Fro}})^2}$$

Returns scalar output.

CHAPTER 3

Indices and Tables

- genindex
- modindex

Python Module Index

t

torch_specinv.methods, 6
torch_specinv.metrics, 8

Index

A

ADMM() (*in module torch_specinv.methods*), 6

G

griffin_lim() (*in module torch_specinv.methods*), 7

L

L_BFGS() (*in module torch_specinv.methods*), 6

P

phase_init() (*in module torch_specinv.methods*), 8

R

RTISI_LA() (*in module torch_specinv.methods*), 7

S

sc() (*in module torch_specinv.metrics*), 8
ser() (*in module torch_specinv.metrics*), 8
snr() (*in module torch_specinv.metrics*), 8

T

torch_specinv.methods (*module*), 6
torch_specinv.metrics (*module*), 8