torch_radon

Release 0.0.1

Matteo Ronchetti

May 18, 2022

GETTING STARTED

Torch Radon is a fast CUDA implementation of transforms needed for working with computed tomography data in Pytorch. It allows the training of end-to-end models that takes sinograms as inputs and produce images as output.

Main features:

- All operations work directly on Pytorch GPU tensors.
- Forward and back projections are differentiable and integrated with Pytorch *.backward()*.
- Up to 50x faster than Astra Toolbox.
- Supports half precision and can used togheter with amp for faster training.

Projection types:

- Parallel Beam
- Fan Beam

CHAPTER ONE

GOOGLE COLAB

The easiest way to start experimenting with the Torch Radon library is to use Google Colab. You can find a sample notebook here.

CHAPTER

TWO

INSTALL LOCALLY

2.1 Precompiled Package

If you are running Linux you can install Torch Radon by running:

```
wget -qO- https://raw.githubusercontent.com/matteo-ronchetti/torch-radon/master/auto_
↪install.py  | python -
```

2.2 Docker Image

Docker images with PyTorch CUDA and Torch Radon are available here.

```
docker pull matteoronchetti/torch-radon
```

To use the GPU in docker you need to use nvidia-docker.

2.3 Compile from Source

You need to have CUDA and PyTorch installed, then run:

```
git clone https://github.com/matteo-ronchetti/torch-radon.git
cd torch-radon
python setup.py install
```

If you encounter any problem please contact the author or open an issue.

CHAPTER

THREE

RADON PROJECTIONS

3.1 Parallel Beam

class `torch_radon.Radon`(*resolution: int*, *angles*, *det_count=- 1*, *det_spacing=1.0*, *clip_to_circle=False*)

Class that implements Radon projection for the Parallel Beam geometry.

Parameters

- **`resolution`** – The resolution of the input images.
- **`angles`** – Array containing the list of measuring angles. Can be a Numpy array or a PyTorch tensor.
- **`det_count`** – Number of rays that will be projected. By default it is = `resolution`
- **`det_spacing`** – Distance between two contiguous rays.
- **`clip_to_circle`** – If True both forward and backward projection will be restricted to pixels inside the circle (highlighted in cyan).

Note: Currently only support resolutions which are multiples of 16.

`forward`(*self*, *x*)

Radon forward projection.

Parameters **`x`** – PyTorch GPU tensor with shape $(d_1, \dots, d_n, r, r)$ where r is the `resolution` given to the constructor of this class.

Returns PyTorch GPU tensor containing sinograms. Has shape $(d_1, \dots, d_n, len(angles), det\_count)$.

`backprojection`(*self*, *sinogram*)

Radon backward projection.

Parameters **sinogram** – PyTorch GPU tensor containing sinograms with shape $(d_1, \dots, d_n, len(angles), det\_count)$.

Returns PyTorch GPU tensor with shape $(d_1, \dots, d_n, r, r)$ where r is the `resolution` given to the constructor of this class.

backward(*self*, *sinogram*)

Same as backprojection

3.2 Fanbeam

class torch_radon.**RadonFanbeam**(*resolution: int*, *angles*, *source_distance: float*, *det_distance: float = - 1*, *det_count: int = - 1*, *det_spacing: float = - 1*, *clip_to_circle=False*)

Class that implements Radon projection for the Fanbeam geometry.

Parameters

- **resolution** – The resolution of the input images.
- **angles** – Array containing the list of measuring angles. Can be a Numpy array or a PyTorch tensor.
- **source_distance** – Distance between the source of rays and the center of the image.
- **det_distance** – Distance between the detector plane and the center of the image. By default it is = `source_distance`.
- **det_count** – Number of rays that will be projected. By default it is = `resolution`.
- **det_spacing** – Distance between two contiguous rays.
- **clip_to_circle** – If True both forward and backward projection will be restricted to pixels inside the circle (highlighted in cyan).

Note: Currently only support resolutions which are multiples of 16.

forward(*self*, *x*)

Radon forward projection.

Parameters **x** – PyTorch GPU tensor with shape $(d_1, \dots, d_n, r, r)$ where r is the `resolution` given to the constructor of this class.

Returns PyTorch GPU tensor containing sinograms. Has shape $(d_1, \dots, d_n, len(angles), det\_count)$.

`backprojection`(*self*, *sinogram*)

Radon backward projection.

Parameters **`sinogram`** – PyTorch GPU tensor containing sinograms with shape $(d_1, \dots, d_n, len(angles), det\_count)$.

Returns PyTorch GPU tensor with shape $(d_1, \dots, d_n, r, r)$ where r is the `resolution` given to the constructor of this class.

`backward`(*self*, *sinogram*)

Same as backprojection

CHAPTER

FOUR

SHEARLET TRANSFORM

class torch_radon.shearlet.**ShearletTransform**(*width*, *height*, *alphas*, *cache=None*)

Implementation of Alpha-Shearlet transform based on https://github.com/dedale-fet/alpha-transform/tree/master/alpha_transform.

Once the shearlet spectrograms are computed all the computations are done on the GPU.

Parameters

- **width** – Width of the images
- **height** – Height of the images
- **alphas** – List of alpha coefficients that will be used to generate shearlets
- **cache** – If specified it should be a path to a directory that will be used to cache shearlet coefficients in order to avoid recomputing them at each instantiation of this class.

Note: Support both float and double precision.

---

forward(*self*, *x*)

Do shearlet transform of a batch of images.

Parameters **x** – PyTorch GPU tensor with shape $(d_1, \ldots, d_n, h, w)$.

Returns PyTorch GPU tensor containing shearlet coefficients. Has shape $(d_1, \ldots, d_n,$

backward(*self*, *cs*)

Do inverse shearlet transform.

Parameters **cs** – PyTorch GPU tensor containing shearlet coefficients, with shape $(d_1, \ldots, d_n,$

CHAPTER

FIVE

SOLVERS

The module `torch_radon.solvers` contains implementations of algorithms that can be used to solve tomographic reconstructions problems.

5.1 Landweber Iteration

class `torch_radon.solvers.`**`Landweber`**(*operator*, *projection=None*, *grad=False*)

Class that implements Landweber iteration to solve $\min_{x \in C} \|Ax - y\|_2^2$ (see Wikipedia page).

The iteration used is $x_{n+1} = \mathcal{P}_C(x - \alpha A^T A x_n)$ where $\mathcal{P}_C$ is the projection onto C.

Parameters

- **`operator`** – Instance of a class that implements products Ax (`operator.forward(x)`) and $A^T y$ (`operator.backward(y)`).
- **`projection`** – Function that implements $\mathcal{P}_C(\cdot)$, if not specified no projection is used.
- **`grad`** – If true gradient will be enabled, more memory will be used but it will be possible to backpropagate.

`estimate_alpha`(*img_size*, *device*, *n_iter=50*, *batch_size=8*)

Use power iteration on $A^T A$ to estimate the maximum step size that still guarantees convergence.

Note: Because this computation is not exact it is advised to use a value of alpha lower that the one estimated by this method (for example multiplying the estimate by 0.95).

Parameters

- **`img_size`** – Size of the image
- **`device`** – GPU device that will be used for computation
- **`n_iter`** – Number of iterations
- **`batch_size`** – Number of vectors used in the power iteration.

Returns Estimated value for alpha

`run`(*x_zero*, *y*, *alpha*, *iterations=100*, *callback=None*)

Execute Landweber iterations.

Parameters

- **x_zero** – Initial solution guess used as a starting point for the iteration
- **y** – Value of y in $\min_{x \in C} \|Ax - y\|_2^2$
- **alpha** – Step size, can be estimated using *estimate_alpha*
- **iterations** – Number of iterations
- **callback** – Optional function that will be called at each iteration with x_n as argument. Values returned by `callback` will be stored in a list and returned together with the computed solution

Returns If `callback` is specified returns `x, values` where `x` is the solution computed by the Landweber iteration and `values` is the list of values returned by `callback` at each iteration. If `callback` is not specified returns only `x`

5.2 Conjugate Gradient

`torch_radon.solvers.cg`(*forward*, *x*, *y*, *callback=None*, *max_iter=500*, *tol=1e-05*)

Implements Conjugate Gradient algorithm for solving $\min_x \|Ax - y\|_2^2$.

Note: For conjugate gradient to work the matrix A must be symmetric positive definite. Otherwise use other solvers.

Parameters

- **forward** – function that implements products Ax (`forward(x)`).
- **x** – Initial solution guess used as a starting point for the iteration
- **y** – Value of y in $\min_{x \in C} \|Ax - y\|_2^2$
- **callback** – Optional function that will be called at each iteration with x_n and the residual as arguments. Values returned by `callback` will be stored in a list and returned together with the computed solution.
- **max_iter** – Maximum number of iterations.
- **tol** – Algorithm is stopped when $\frac{\|Ax_n - y\|}{\|y\|} \leq \text{tol}$

Returns If `callback` is specified returns `x, values` where `x` is the solution computed by the Landweber iteration and `values` is the list of values returned by `callback` at each iteration. If `callback` is not specified returns only `x`.

`torch_radon.solvers.cgne`(*operator*, *x*, *y*, *callback=None*, *max_iter=5000*, *tol=1e-05*)

Implements Conjugate Gradient on the Normal Equations, an algorithm for solving $\min_x \|Ax - y\|_2^2$.

Parameters

- **operator** – Instance of a class that implements products Ax (`operator.forward(x)`) and $A^T y$ (`operator.backward(y)`).
- **x** – Initial solution guess used as a starting point for the iteration :param y: Value of y in $\min_{x \in C} \|Ax - y\|_2^2$
- **callback** – Optional function that will be called at each iteration with x_n as argument. Values returned by `callback` will be stored in a list and returned together with the computed solution

- **max_iter** – Maximum number of iterations
- **tol** – Algorithm is stopped when $\frac{||s||}{||y||} \le \text{tol}$

Returns If `callback` is specified returns `x, values` where `x` is the solution computed by the Landweber iteration and `values` is the list of values returned by `callback` at each iteration. If `callback` is not specified returns only `x`

CHAPTER

SIX

INDICES AND TABLES

- genindex

INDEX