trixi Documentation

Release 0.1

MIC

Jun 14, 2019

Contents

1 Quick Start 3

2 License 5

3 Authors 7

4 trixi.experiment 9

5 trixi.experiment_browser 13

6 trixi.logger 15

7 trixi.util 19

8 Class Diagram 21

9 Indices and tables 23

Python Module Index 25

Index 27

This is the documentation of **trixi** (Training & Retrospective Insights eXperiment Infrastructure). **trixi** is a python package aiming to facilitate the setup, visualization and comparison of reproducible experiments, currently with a focus on experiments using PyTorch.

You can jump right into the package by looking into our *Quick Start*.

Install Trixi:

```
pip install trixi
```

Install trixi directly via git:

```
git clone https://github.com/MIC-DKFZ/trixi.git
cd trixi
pip install -e .
```

CHAPTER 1

Quick Start

Introduction & Features:

https://github.com/MIC-DKFZ/trixi#features

Install trixi:

```
pip install trixi
```

Have a look and run a simple MNIST example:

https://github.com/MIC-DKFZ/trixi/blob/master/examples/pytorch_experiment.ipynb

CHAPTER 2

License

```
The MIT License (MIT)

Copyright (c) 2018 Medical Image Computing Group, DKFZ

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER 3

Authors

Core Development Team:

- David Zimmerer (d.zimmerer@dkfz.de)
- Jens Petersen (jens.petersen@dkfz.de)
- Gregor Koehler (g.koehler@dkfz.de)

Contributions:

- Jakob Wasserthal
- Sebastian Wirkert
- Lisa Kausch

CHAPTER 4

trixi.experiment

4.1 Experiment

class trixi.experiment.experiment.**Experiment**(*n_epochs=0*)

Bases: object

An abstract Experiment which can be run for a number of epochs.

The basic life cycle of an experiment is:

```
setup()
prepare()

while epoch < n_epochs:
    train()
    validate()
    epoch += 1

end()
```

If you want to use another criterion than number of epochs, e.g. stopping based on validation loss, you can implement that in your validation method and just call .stop() at some point to break the loop. Just set your n_epochs to a high number or np.inf.

The reason there is both *setup()* and *prepare()* is that internally there is also a _setup_internal() method for hidden magic in classes that inherit from this. For example, the trixi.experiment.pytorchexperiment.PytorchExperiment uses this to restore checkpoints. Think of *setup()* as an __init__() that is only called when the Experiment is actually asked to do anything. Then use *prepare()* to modify the fully instantiated Experiment if you need to.

To write a new Experiment simply inherit the Experiment class and overwrite the methods. You can then start your Experiment calling *run()*

In Addition the Experiment also has a test function. If you call the *run_test()* method it will call the *test()* and *end_test()* method internally (and if you give the parameter setup = True in run_test is will again call *setup()* and *prepare()*).

Each Experiment also has its current state in `_exp_state`, its start time in `_time_start`, its end time in `_time_end` and the current epoch index in `_epoch_idx`

Parameters **n_epochs** (*int*) – The number of epochs in the Experiment (how often the train and validate method will be called)

end()
Is called at the end of each experiment

end_test()
Is called at the end of each experiment test

epoch
Convenience access property for self._epoch_idx

prepare()
This method is called directly before the experiment training starts

process_err(*e*)
This method is called if an error occurs during the execution of an experiment. Will just raise by default.

Parameters **e** (*Exception*) – The exception which was raised during the experiment life cycle

run(*setup=True*)
This method runs the Experiment. It runs through the basic lifecycle of an Experiment:

```
setup()
prepare()

while epoch < n_epochs:
    train()
    validate()
    epoch += 1

end()
```

run_test(*setup=True*)
This method runs the Experiment.

The test consist of an optional setup and then calls the *test()* and *end_test()*.

Parameters **setup** – If True it will execute the *setup()* and *prepare()* function similar to the run method before calling *test()*.

setup()
Is called at the beginning of each Experiment run to setup the basic components needed for a run.

stop()
If called the Experiment will stop after that epoch and not continue training

test()
The testing part of the Experiment

train(*epoch*)
The training part of the Experiment, it is called once for each epoch

Parameters **epoch** (*int*) – The current epoch the train method is called in

validate(*epoch*)
The evaluation/validation part of the Experiment, it is called once for each epoch (after the training part)

Parameters **epoch** (*int*) – The current epoch the validate method is called in

4.2 PytorchExperiment

CHAPTER 5

trixi.experiment_browser

5.1 browser

5.2 dataprocessing

5.3 ExperimentReader

CHAPTER 6

trixi.logger

6.1 experiment

6.1.1 ExperimentLogger

6.1.2 PytorchExperimentLogger

6.2 file

6.2.1 NumpyPlotFileLogger

6.2.2 PytorchPlotFileLogger

6.2.3 TextFileLogger

6.3 message

6.3.1 SlackMessageLogger

6.3.2 TelegramMessageLogger

6.4 plt

6.4.1 NumpySeabornPlotLogger

6.4.2 NumpySeabornImagePlotLogger

6.5 tensorboard

6.5.1 TensorboardXLogger

6.5.2 PytorchTensorboardXLogger

Abstract interface for visual logger.

process_params(*f*, **args*, ***kwargs*)
Implement this to handle data conversions in your logger.

Example: Implement logger for numpy data, then implement torch logger as child of numpy logger and just use the process_params method to convert from torch to numpy.

show_barplot(**args*, ***kwargs*)
Abstract method which should handle and somehow log/ store a barplot

show_image(**args*, ***kwargs*)
Abstract method which should handle and somehow log/ store an image

show_lineplot(**args*, ***kwargs*)
Abstract method which should handle and somehow log/ store a lineplot

show_piechart(**args*, ***kwargs*)
Abstract method which should handle and somehow log/ store a piechart

show_scatterplot(**args*, ***kwargs*)
Abstract method which should handle and somehow log/ store a scatterplot

show_text(**args*, ***kwargs*)
Abstract method which should handle and somehow log/ store a text

show_value(**args*, ***kwargs*)
Abstract method which should handle and somehow log/ store a value

trixi.logger.abstractlogger.**convert_params**(*f*)
Decorator to call the process_params method of the class.

trixi.logger.abstractlogger.**threaded**(*f*)
Decorator to run the process in an extra thread.

6.8 CombinedLogger

class trixi.logger.combinedlogger.**CombinedLogger**(**loggers*)
Bases: object

A Logger which can combine all other logger and if called calls all the sub loggers

trixi.logger.combinedlogger.**create_function**(*self*, *sub_methods*)

CHAPTER 7

trixi.util

7.1 Config

7.2 ExtraVisdom

7.3 GridSearch

7.4 pytorchutils

7.5 SourcePacker

CHAPTER 8

Class Diagram

8.1 Logger

8.2 Experiment

CHAPTER 9

Indices and tables

- genindex
- modindex

Python Module Index

t

trixi.experiment, 9
trixi.experiment.experiment, 9
trixi.logger.abstractlogger, 16
trixi.logger.combinedlogger, 17

Index

A

AbstractLogger (*class in trixi.logger.abstractlogger*), 16

C

CombinedLogger (*class in trixi.logger.combinedlogger*), 17
convert_params() (*in module trixi.logger.abstractlogger*), 17
create_function() (*in module trixi.logger.combinedlogger*), 17

E

end() (*trixi.experiment.experiment.Experiment method*), 10
end_test() (*trixi.experiment.experiment.Experiment method*), 10
epoch (*trixi.experiment.experiment.Experiment attribute*), 10
Experiment (*class in trixi.experiment.experiment*), 9

P

prepare() (*trixi.experiment.experiment.Experiment method*), 10
process_err() (*trixi.experiment.experiment.Experiment method*), 10
process_params() (*trixi.logger.abstractlogger.AbstractLogger method*), 17

R

run() (*trixi.experiment.experiment.Experiment method*), 10
run_test() (*trixi.experiment.experiment.Experiment method*), 10

S

setup() (*trixi.experiment.experiment.Experiment method*), 10
show_barplot() (*trixi.logger.abstractlogger.AbstractLogger method*), 17
show_image() (*trixi.logger.abstractlogger.AbstractLogger method*), 17
show_lineplot() (*trixi.logger.abstractlogger.AbstractLogger method*), 17
show_piechart() (*trixi.logger.abstractlogger.AbstractLogger method*), 17
show_scatterplot() (*trixi.logger.abstractlogger.AbstractLogger method*), 17
show_text() (*trixi.logger.abstractlogger.AbstractLogger method*), 17
show_value() (*trixi.logger.abstractlogger.AbstractLogger method*), 17
stop() (*trixi.experiment.experiment.Experiment method*), 10

T

test() (*trixi.experiment.experiment.Experiment method*), 10
threaded() (*in module trixi.logger.abstractlogger*), 17
train() (*trixi.experiment.experiment.Experiment method*), 10
trixi.experiment (*module*), 9
trixi.experiment.experiment (*module*), 9
trixi.logger.abstractlogger (*module*), 16
trixi.logger.combinedlogger (*module*), 17

V

validate() (*trixi.experiment.experiment.Experiment method*), 10