chainerui Documentation

Release 0.11.0

Preferred Networks, inc.

Dec 26, 2019

ChainerUI Documents

CHAPTER 1

Installation Guide

1.1 Dependencies

ChainerUI is developed under Python 2.7+, 3.5+, 3.6+, 3.7+. For other requirements, see `requirements.txt`.

Listing 1: requirements.txt

```
enum34>=1.1.6; python_version < '3.4'
msgpack>=0.5.6
Flask>=1.1.0
sqlalchemy>=1.1.18
alembic>=1.0.0
gevent>=1.2.2
structlog>=18.2.0
filelock>=3.0.9
urllib3>=1.24.1
```

ChainerUI uses `sqlite3` module which is included in the standard Python library. If Python is built from source, `sqlite3` must be installed before building Python.

- On Ubuntu, `libsqlite3-dev` must be installed before building Python (`$ apt-get install libsqlite3-dev`).
- On Windows, install Visual C++ Build Tools with the Default Install setting before building Python.

1.2 Install ChainerUI

1.2.1 Install ChainerUI via PyPI

To install ChainerUI, use `pip`:

```
$ pip install chainerui
```

1.2.2 Install ChainerUI from source

To install ChaineruI from source, build from a cloned Git repository. Frontend module requires `npm` 6.2.0+:

```
$ git clone https://github.com/chainer/chainerui.git
$ cd chainerui/frontend
$ npm install && npm run build && cd ..
$ pip install -e .
```

1.3 Quick start

Initialize ChainerUI database:

```
$ chainerui db create
$ chainerui db upgrade
```

Clone examples of train log and create a project:

```
$ git clone https://github.com/chainer/chainerui.git
$ cd chainerui

$ # create example project project
$ chainerui project create -d examples/log-file -n example-project
```

Run ChainerUI server:

```
$ chainerui server
```

Open http://localhost:5000/ and select "example-project", then shown a chart of training logs.

For more detailed usage, see *Getting started*.

1.4 Docker start

Get Docker container from DockerHub and start ChainerUI server. The container has installed ChainerUI module, setup a DB and a command to start the server:

```
$ git clone https://github.com/chainer/chainerui.git
$ cd chainerui

$ # replace tag to the latest version number
$ docker pull chainer/chainerui:latest
$ docker run -d -p 5000:5000 --name chainerui chainer/chainerui:latest
$ # then ChainerUI server is running
```

Open http://localhost:5000/ shown empty project list.

Form more detailed usage, see *Getting started* or *Use web API*.

1.5 Browser compatibility

ChainerUI is supported by the latest stable version of the following browsers.

- Firefox
- Chrome

CHAPTER 2

Getting started

Output `log` files during training phase, and ChainerUI collects them. When send training logs via web API, see *web API* section.

2.1 Create a database

Please setup database at first:

```
$ chainerui db create
$ chainerui db upgrade
```

2.2 Create a project

```
$ chainerui project create -d PROJECT_DIR [-n PROJECT_NAME]
```

The ChainerUI server watches the files below the project directory recursively.

- `log`: Used for chart.
- `args`: (optional) Used for *result table*, show as experimental conditions.
- `commands`: (optional) Created by *CommandsExtension* internally, used for operating training job.

For more detail of the files and how to setup training loop, see *Customize training loop*

For example, look at the file and directory structure below. When create a project with `-d path/to/result`, the results of the two directories, `result1` and `result2` are registered under the `PROJECT_DIR` (or `PROJECT_NAME`) automatically, then ChainerUI continuously gathers the both logs.:

```
path/to/result/result1
  |--- log       # show values on chart
  |--- args      # show parameters on result table as experimental conditions
```

(continues on next page)

(continued from previous page)

```
  |--- commands  # created by CommandsExtension to operate the training loop
  |--- ...
path/to/result/result2
  |--- log
  |--- args
  |--- commands
  |--- ...
```

2.3 Start ChainerUI server

```
$ chainerui server
```

Open http://localhost:5000/ . To stop, press `Ctrl+C` on the console. When use original host or port, see *command option*:

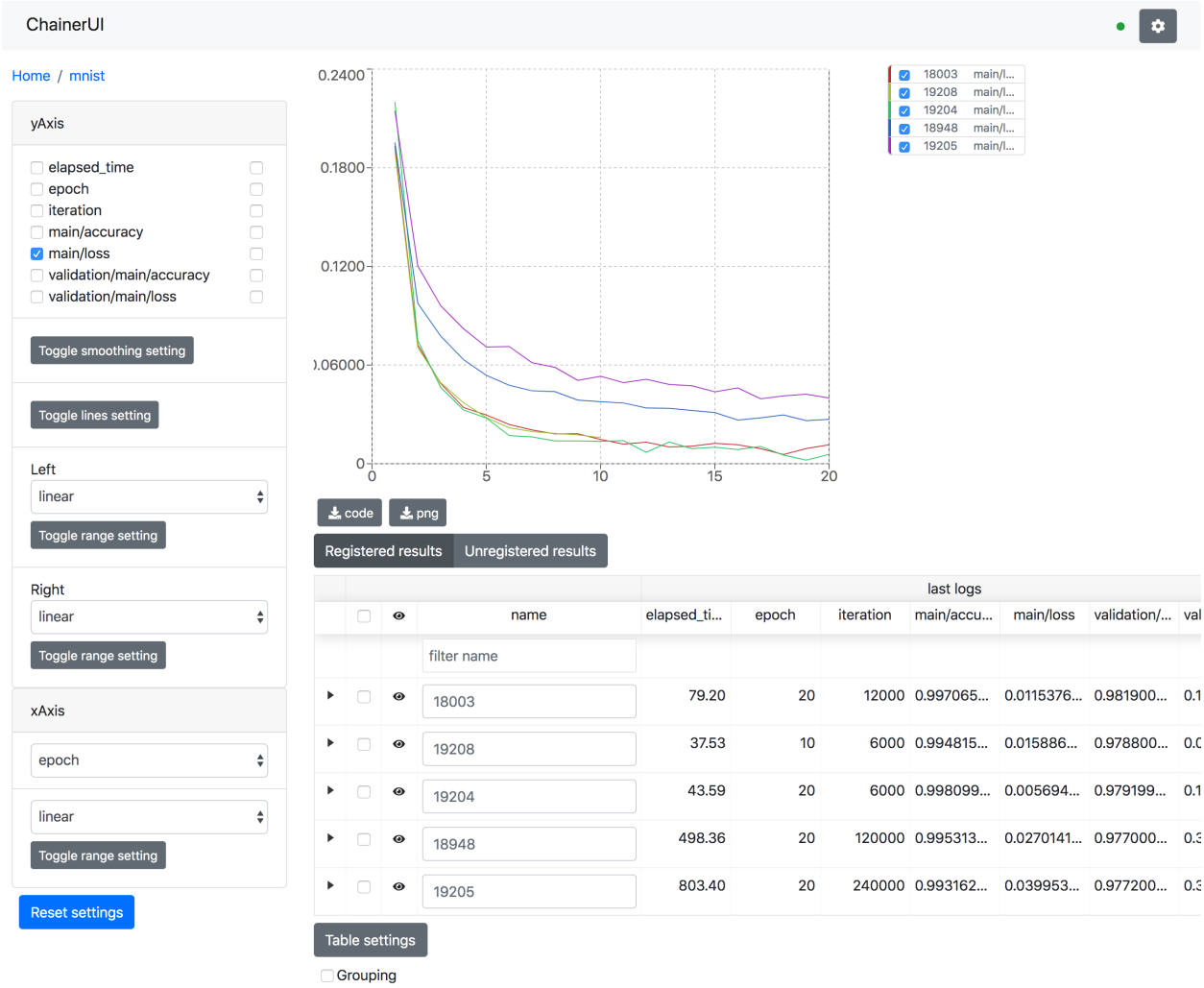
2.4 Customize training loop

ChainerUI basically supports the Trainer module included in Chainer, and some functions without `Trainer`.

Note: examples/log-file/train_mnist.py, based on chainer/examples/mnist/train_mnist.py, is a useful example to see how to set training loops with ChainerUI.

Note: examples/log-file/train_mnist_custom_loop.py is an example, basaed on chainer/examples/mnist/train_mnist_custom_loop, which does not use the training loop from `Trainer`. However, this example will not use the training loop from *Operate training loop*.

2.4.1 Training log



ChainerUI plots training log values read from the `log` files and shows the training job. The `log` file is a JSON file created by LogReport extension or *chainerui's LogReport*, which is registered automatically and created under the project path. If `log` files are updated, the chart and results table are also updated continuously.

Note: `epoch`, `iteration`, `episode`, `step` and `elapsed_time` are assumed as x-axis. X-axis of a chart is selected by `xAxis` pane.

- LogReport extension sets `epoch`, `iteration` and `elapsed_time` automatically.
- *chainerui's LogReport* sets `elapsed_time` automatically. Other x-axis keys have to be set manually if necessary.

Note: When retrying a training job with a same directory, `log` file will be truncated and created, then the job overwrites logs the file. But ChainerUI cannot distinguish whether the `log` file is updated or recreated. ChainerUI recommends to create another directory for output result on retrying.

Setup example from a brief MNIST example:

```
import chainer.links as L
from chainer import training
from chainer.training import extensions

def main():
    # Classifier reports softmax cross entropy loss and accuracy at every
    # iteration
    # [ChainerUI] plot loss and accuracy reported by this link
    model = L.Classifier(MLP(args.unit, 10))

    trainer = training.Trainer(updater, (args.epoch, 'epoch'), out=args.out)

    # [ChainerUI] read 'log' file for plotting values
    trainer.extend(extensions.LogReport())
```

Created `log` file example:

```
[
    {
        "main/loss": 0.1933198869228363,
        "validation/main/loss": 0.09147150814533234,
        "iteration": 600,
        "elapsed_time": 16.052587032318115,
        "epoch": 1,
        "main/accuracy": 0.9421835541725159,
        "validation/main/accuracy": 0.9703000783920288
    },
    {
        "main/loss": 0.07222291827201843,
        "validation/main/loss": 0.08141259849071503,
        "iteration": 1200,
        "elapsed_time": 19.54666304588318,
        "epoch": 2,
        "main/accuracy": 0.9771820902824402,
        "validation/main/accuracy": 0.975399911403656
    },
    ...
]
```

A example without `Trainer` code, from a short extract of the MNIST custom loop example:

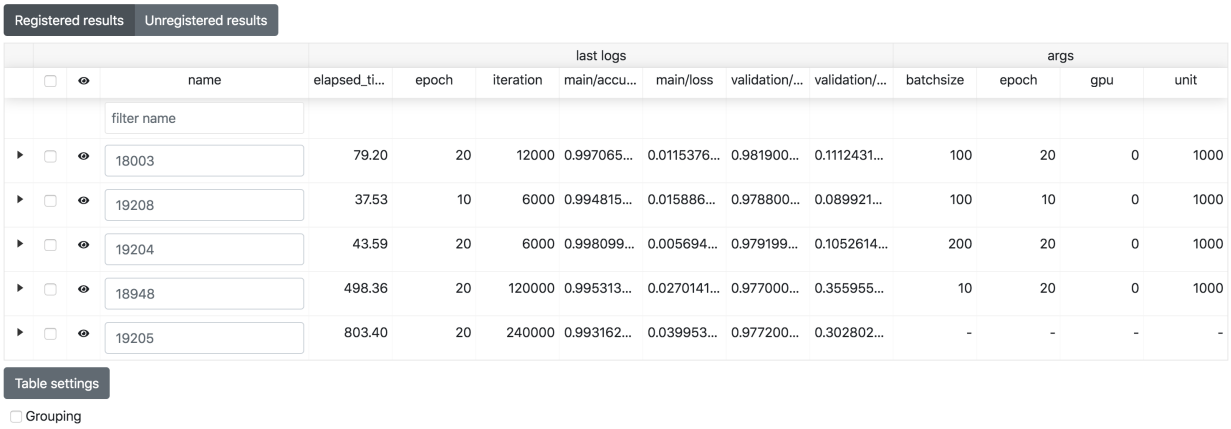
```
from chainerui.utils import LogReport

def main():

    # [ChainerUI] setup log reporter to show on ChainerUI along with 'args'
    ui_report = LogReport(args.out, conditions=args)
    while train_iter.epoch < args.epoch:

        # ...train calculation

        if train_iter.is_new_epoch:

            # [ChainerUI] write values to 'log' file
            stats = {
                'epoch': train_iter.epoch,
                'iteration': train_iter.epoch * args.batchsize,
```

(continues on next page)

(continued from previous page)

```
                'train/loss': train_loss, 'train/accuracy': train_accuracy,
                'test/loss': test_loss, 'test/accuracy': test_accuracy
                }
            ui_report(stats)
```

2.4.2 Experimental conditions



ChainerUI shows the training job with experimental conditions read from the `args` file. `args` file is a JSON file, which includes key-value pairs. See *save_args*, util function to dump command line arguments or dictionaries to `args` file.

Setup example of a brief MNIST example:

```
# [ChainerUI] import chainerui util function
from chainerui.utils import save_args

def main():
    parser.add_argument('--out', '-o', default='result',
                        help='Directory to output the result')
    args = parser.parse_args()

    # [ChainerUI] save 'args' to show experimental conditions
    save_args(args, args.out)
```

Here is an `args` file examples, with values shown as experimental conditions on a results table:

```
{
    "resume": "",
    "batchsize": 100,
    "epoch": 20,
    "frequency": -1,
    "gpu": 0,
    "unit": 1000,
    "out": "results"
}
```

2.4.3 Operate training loop

ChainerUI supports operating a training loop with *CommandsExtension*. The latest version supports:

- Taking snapshot
- Adjusting the hyperparameters of an optimizer
- Stopping the training loop

Operation buttons are in result table row, click button to expand, or in *result page*, click `Detail` button.

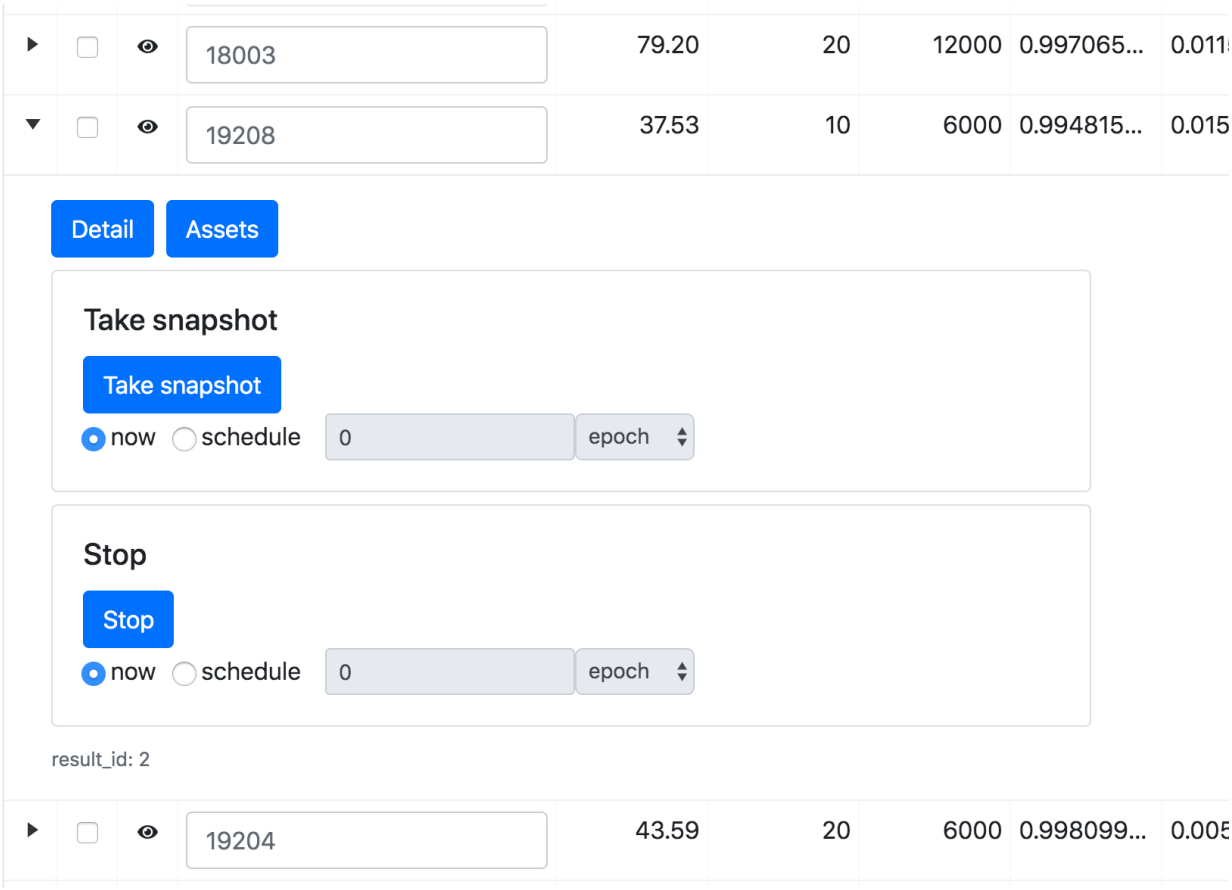


Fig. 1: expand table row to show sub components.

Setup example of a brief extract MNIST example:

```
from chainer import training
from chainer.training import extensions

# [ChainerUI] import CommandsExtension
from chainerui.extensions import CommandsExtension

def main():
    trainer = training.Trainer(updater, (args.epoch, 'epoch'), out=args.out)

    # [ChainerUI] Observe learning rate
    trainer.extend(extensions.observe_lr())
```

(continues on next page)

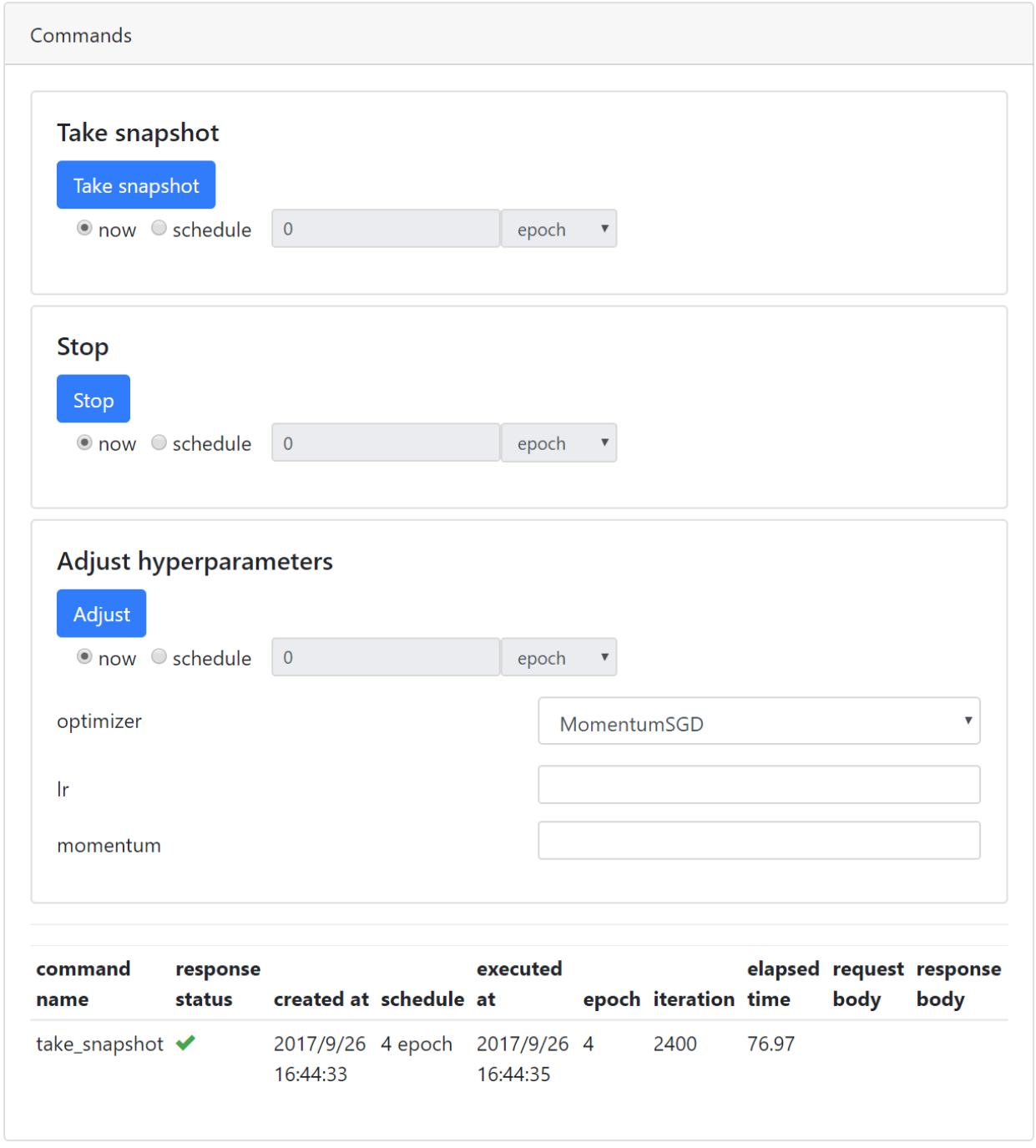


Fig. 2: commands pane of result page

(continued from previous page)

```
    # [ChainerUI] enable to send commands from ChainerUI
    trainer.extend(CommandsExtension())
```

Note: This operation of a training loop is from the *CommandsExtension* which requires `Trainer`. A training loop without `Trainer` cannot use this function.

Note: Adjusting the hyperparameters supports only MomentumSGD and learning rate (`lr`). The optimizer is required to be registered by the name `'main'`.

Support

```
updater = training.StandardUpdater(train_iter, optimizer, device=args.gpu)
```

```
updater = training.StandardUpdater(train_iter, {'main': optimizer}, device=args.gpu)
```

Not support

```
updater = training.StandardUpdater(train_iter, {'sub': optimizer}, device=args.gpu)
```

CHAPTER 3

Use web API

Send training logs via web API.

3.1 Start ChainerUI server

```
$ chainerui server
```

Open http://localhost:5000/ . To stop, press `Ctrl+C` on the console. When use original host or port, see *command option*.

Or, use ChainerUI's docker container to run ChainerUI server, see *docker start*.

3.2 Customize training loop

Setup example from a brief MNIST example:

```
import chainerui

def main():
    args = parser.parse_args()

    # [ChainerUI] To use ChainerUI web client, must initialize
    # args will be shown as parameter of this experiment.
    chainerui.init(conditions=args)

    # Set up a neural network to train
    # Classifier reports softmax cross entropy loss and accuracy at every
    # iteration, which will be used by the PrintReport extension below.
    # [ChainerUI] plot loss and accuracy reported by this link
    model = L.Classifier(MLP(args.unit, 10))
```

(continues on next page)

(continued from previous page)

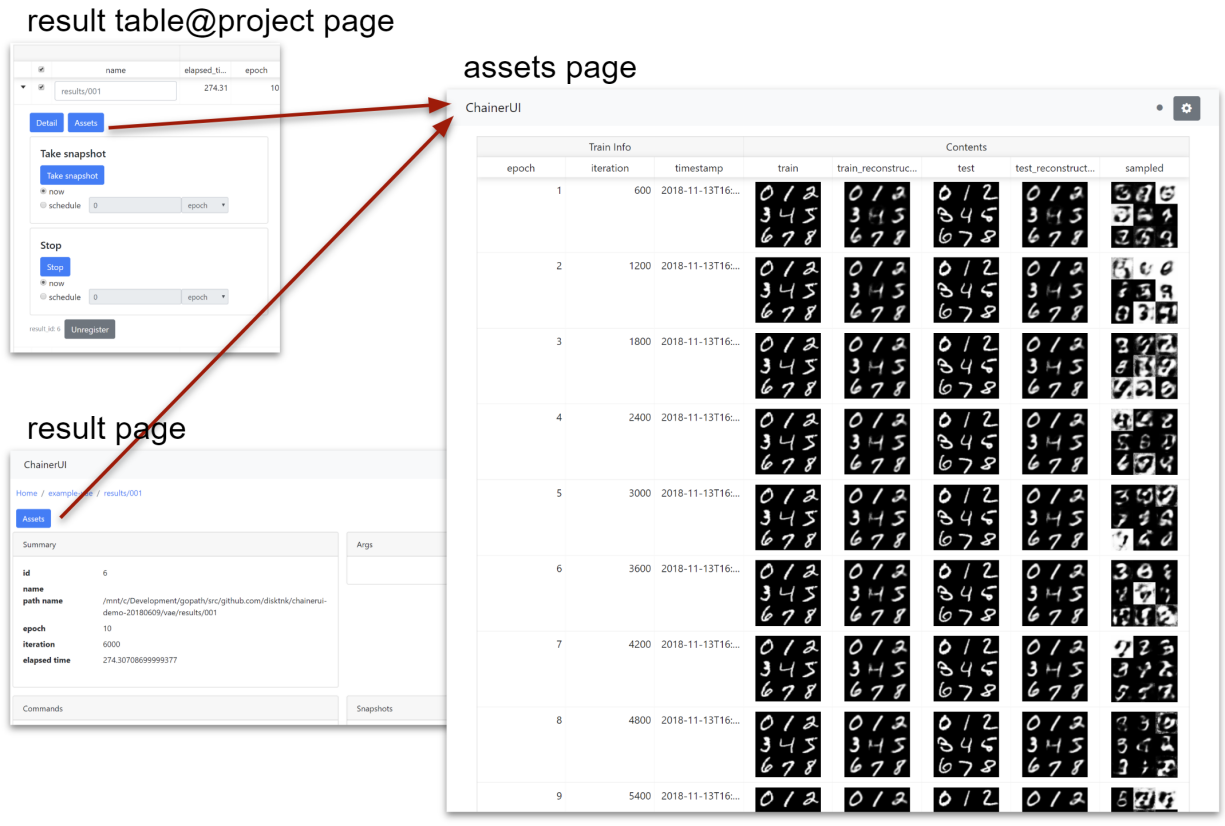
```
    trainer = training.Trainer(updater, (args.epoch, 'epoch'), out=args.out)

    # [ChainerUI] set log reporter on the extention
    trainer.extend(extensions.LogReport(
        postprocess=chainerui.log_reporter()))
```

Note: User doesn't have to execute `$ chainerui project create` command. `chainerui.init()` add a project using current directory on the first running. Project name can be customized using `project_name` option. Training results wil be created every running. Result name is set timestamp automatically and can be customized via web UI.

CHAPTER 4

Visualize assets

ChainerUI provides `/assets` endpoint from v0.8.0 to visualize media assets such image or audio. Basically by using *Asset summaries* module, functions convert `ndarray` to the specified media type or collect texts, as it turns out to show them on a web browser. Assets page can be seen from `assets` button on *result table* or *result detail*.



Note: `chainerui.summary` module requires output directory path. The path **must be same as the directory put**

`log` **file** to gather media assets as experimental result. The `log` file is created by LogReport extension or *chainerui's LogReport*.

```
from chainer import training
from chainerui import summary

out_put = '/path/to/result/'
trainer = training.Trainer(updater, out=out_put)
trainer.extend(training.extensions.LogReport())  # log file will be created at `out_
↪put`
summary.set_out(out_put)  # set output directory as global
```

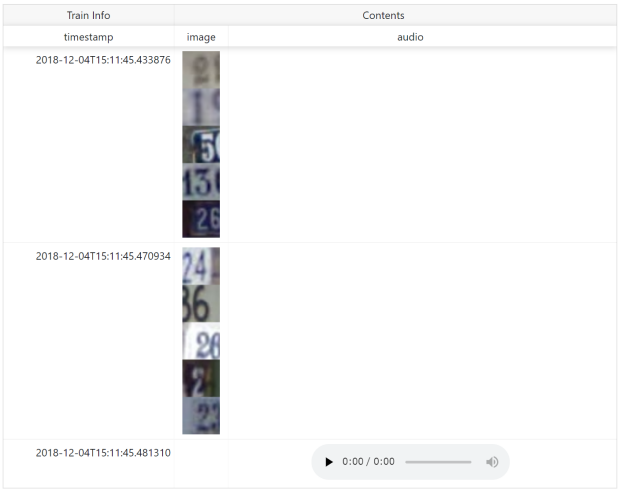
From next section, example codes are skipped getting assets and set output directory, like blow snippet.

```
# get color images and grayscale images for example
import chainer
images, _ = chainer.datasets.get_svhn(withlabel=False)
images_gs, _ = chainer.datasets.get_mnist(withlabel=False, ndim=2)

# make dummy audio data for example
import numpy
audio = numpy.random.uniform(-1, 1, 16000)

from chainerui import summary
summary.set_out('/path/to/result')
```

4.1 Use summary function

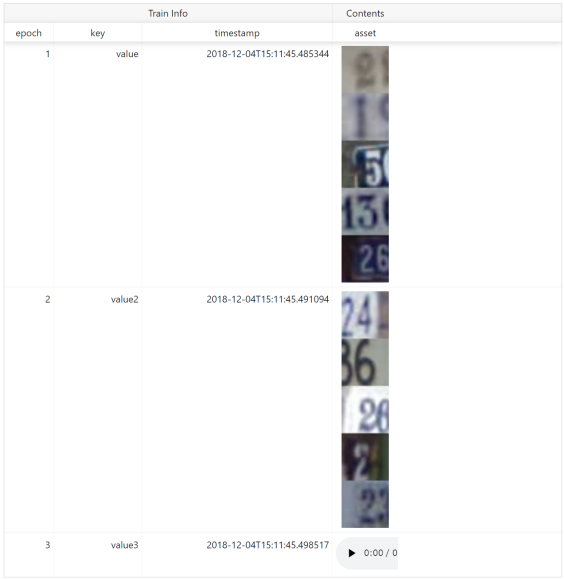
There are 2 ways to show assets on a web browser. First, use `summary` module function directly. The below example is a simple code to show images.

```
summary.image(images[0:5])
summary.image(images[5:10])
summary.audio(audio, 16000)
```



`name` is shown as column name. When show assets with additional text information such as epoch number, iteration number, descriptions and so on, add them as `**kwargs`.

```
summary.image(images[0:5], name='asset', epoch=1, key='value')
summary.image(images[5:10], name='asset', epoch=2, key='value2')
summary.audio(audio, 16000, name='asset', epoch=3, key='value3')
```



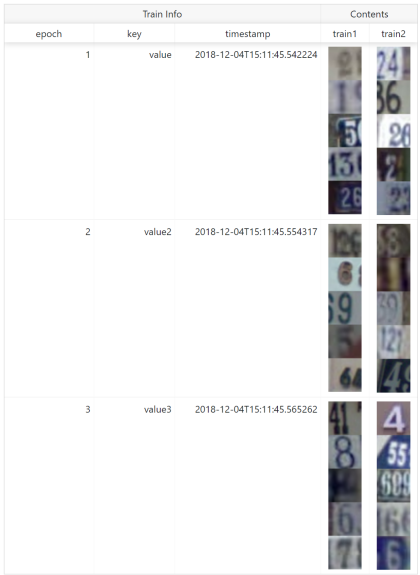
4.2 Use `reporter` function

Second, to aggregate assets to show them in a same row, use *reporter* function. Assets called under `with` statement are aggregated.

```
with summary.reporter() as r:
    r.image(images[0:5])
    r.image(images[5:10])
with summary.reporter() as r:
    r.image(images[10:15])
    r.image(images[15:20])
with summary.reporter() as r:
    r.image(images[20:25])
    r.image(images[25:30])
```

`name` is shown as column name. `reporter` also supports `**kwargs` to add other text information.

```
with summary.reporter(epoch=1, key='value') as r:
    r.image(images[0:5], name='train1')
    r.image(images[5:10], name='train2')
with summary.reporter() as r:
    r.image(images[10:15], name='train1')
    r.image(images[15:20], name='train2')
with summary.reporter() as r:
    r.image(images[20:25], name='train1')
    r.image(images[25:30], name='train2')
```

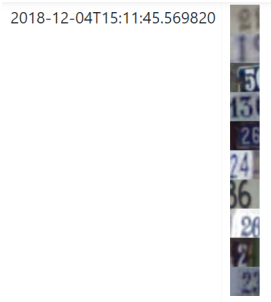


4.3 Image

Required Pillow to use this function.

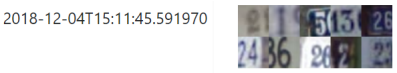
Convert `ndarray` to image as PNG format, save, and report to ChainerUI server. `image` function has some options to customize showing.

- Channel position: Dimensions of `ndarray` is considered as batch, channel, height, width on default. If channel is not in 2nd (= `[1]` in 0-origin) dimension, set `ch_axis` option. For example `ndarray` are batch, height, width, channel order, set `ch_axis=-1`.
- Batched or not: Images are considered as bathed array on default. If an array is not batched, set `batched=False`.
- Tiled: Batched array is showed in one line on default. If show tiled them, set `row` option. For example, batch size is 20 and set `row=4`, images are tiled 4x5 on web browser.
- Color space: If images are not `RGB` or `RGBA` color model, set the color mode with `mode` option. ChainerUI support `HSV` color model, set `mode='HSV'`.

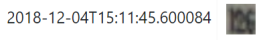


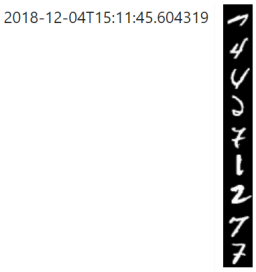
- `image(images[0:10])`:
- `image(images[0:10], row=1)`:

- `image(images[0:10], row=2)`:



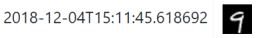
- `image(images[10], ch_axis=0, batched=False)`:





- `image(images_gs[0:9])`:
- `image(images_gs[0:9], row=1)`:

- `image(images_gs[0:9], row=3)`:
- `image(images_gs[9], batched=False)`:



4.4 Audio

Required Scipy to use this function.

Convert `ndarray` to audio as WAV format, save and report to ChainerUI server. `audio` function requires sample rate.

4.5 Text

Simply collect texts, save and report to ChainerUI server.

CHAPTER 5

Use external database

ChainerUI provides `--db` option and supports `CHAINERUI_DB_URL` variable to use external database instead of ChainerUI's default database. Sub-commands, `db`, `project` and `server` look up a value of the database URL in the following order.

1. command option: `--db`
2. environment variable: `CHAINERUI_DB_URL`
3. default database

In the below commands, for example, ChainerUI use `ANOTHER_DB`:

```
$ export CHAINERUI_DB_URL=YOUR_DB
$ chainerui --db ANOTHER_DB server
$ # the server will run with ANOTHER_DB, not use YOUR_DB
```

Note: On default, ChainerUI uses SQLite. The database file is placed at `~/.chainerui/db`.

Note: If use external database, `chainerui db create` is not required for setup.

Supported database types depend on SQLAlchemy, please see Dialect section and setup appropriate driver for the database. The following sections are examples to setup database and connect with them.

Note:

`--db` option value have to be set on each `db`, `project` and `server` sub-commands when use external database:

```
$ chainerui --db YOUR_DB db upgrade

$ # chainerui project create -d PROJECT_DIR # <- *NOT* use YOUR_DB
```

(continues on next page)

(continued from previous page)

```
$ chainerui --db YOUR_DB project create -d PROJECT_DIR

$ # chainerui server  # <- *NOT* use YOUR_DB
$ chainerui --db YOUR_DB server
```

On the other hand, once `CHAINERUI_DB_URL` is set as environment variable, the database URL is shared between other sub-commands.

5.1 Example: SQLite

When use SQLite with an original database file placed at `/path/to/original.db`, database URL is `sqlite:///path/to/original.db`:

```
$ export CHAINERUI_DB_URL=sqlite:////path/to/original.db
$ chainerui db upgrade
$ chainerui server
```

5.2 Example: PostgreSQL

The below example uses `psycopg2` and `postgres:10.5` docker image:

```
$ docker pull postgres:10.5
$ docker run --name postgresql -p 5432:5432 -e POSTGRES_USER=user -e POSTGRES_
↪PASSWORD=pass -d postgres:10.5

$ pip install psycopg2-binary
$ export CHAINERUI_DB_URL=postgresql://user:pass@localhost:5432
$ chainerui db upgrade
$ chainerui server
```

5.3 Example: MySQL

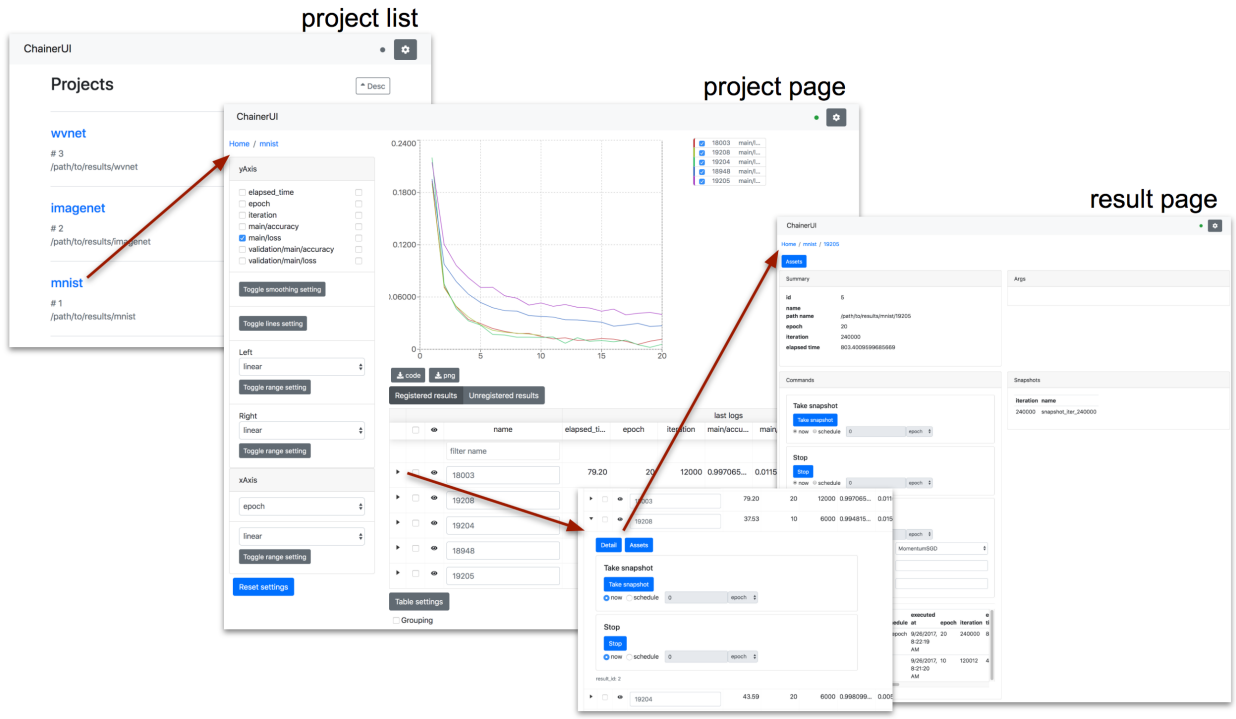
The below example uses `mysqlclient` and `mysql:8.0.12` docker image:

```
$ docker pull mysql:8.0.12
$ docker run --name mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root_pass -e MYSQL_
↪USER=user -e MYSQL_PASSWORD=pass -e MYSQL_DATABASE=chainerui -d mysql:8.0.12

$ pip install mysqlclient
$ export CHAINERUI_DB_URL=mysql+mysqldb://user:pass@127.0.0.1:3306/chainerui
$ chainerui db upgrade
$ chainerui server
```

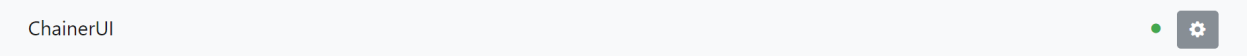
CHAPTER 6

User interface manual

Page transition flow:

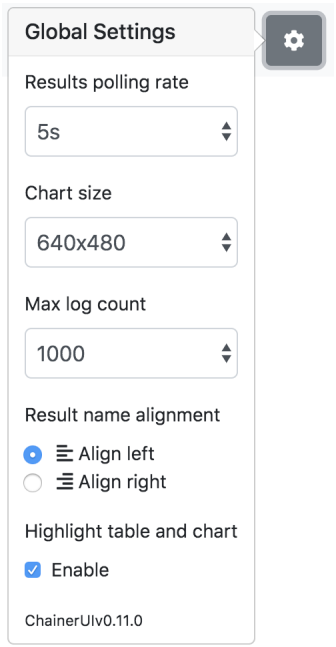


6.1 Header



-  : setup global settings and show ChainerUI version. See *Global settings* section below for more details.
- [connection status between ChainerUI server]
  - green: success to connect
  - blue: loading
  - red: fail to connect
  - gray: disable polling

6.2 Global settings



Results polling rate

Results polling rate is intervals between updates of results on project pages. When you feel your browser is slow, try choosing a longer value.

Chart size

Chart size is the size of the main plot on project pages.

Max log count

Max log count is the maximum number of logs per result that the ChainerUI server sends to the browser on each request. When you feel your browser is slow, try choosing a smaller value.

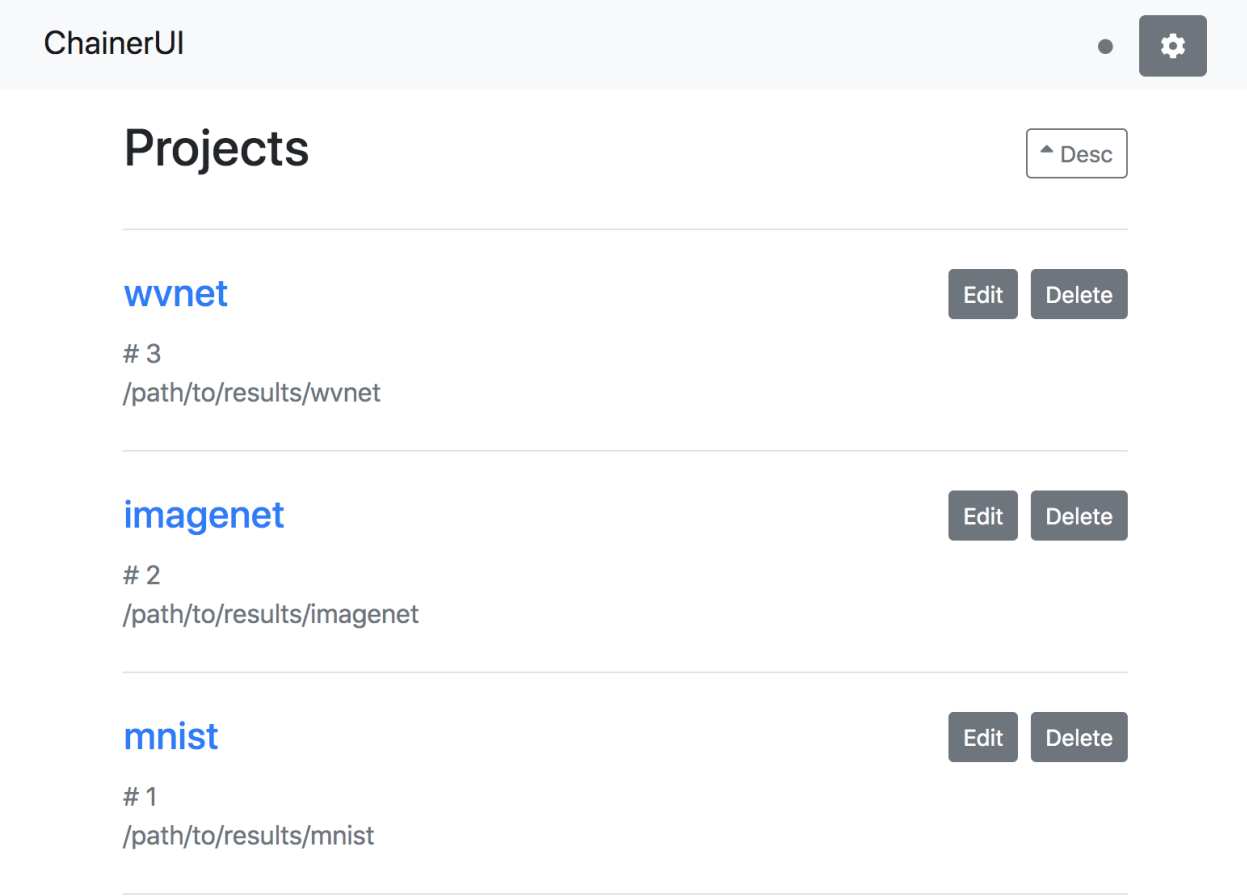
Result name alignment

Result name alignment controls which side of a result name to be truncated when it is too long to be displayed.

Highlight table and chart

Enable highlighting linked betwheen a table row and a log chart. `Enabled` on default.

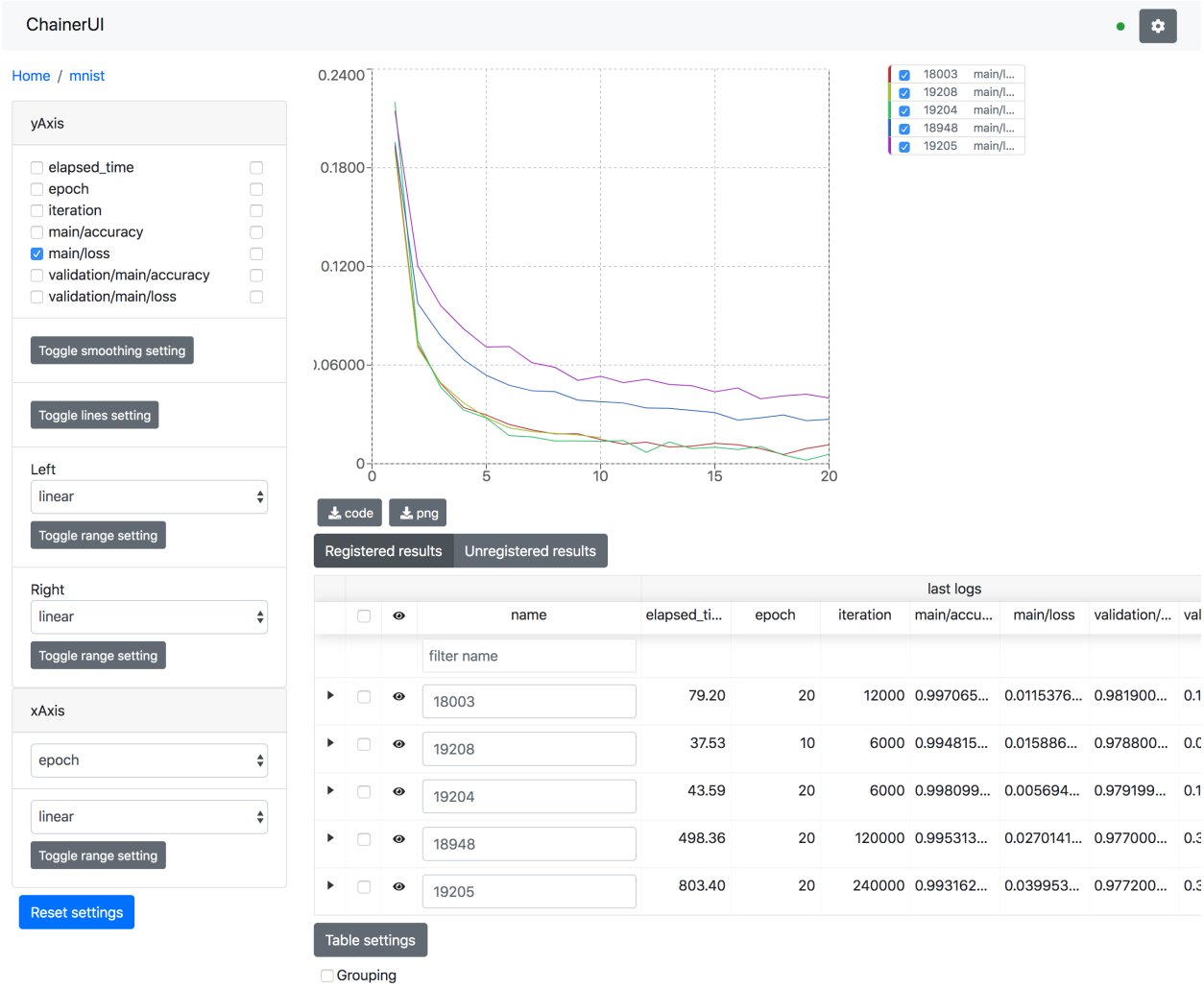
6.3 Home: Project list



From the list of registered projects, select a project to transition to the project page. When registering a project within running server, refresh the page and it will show the project on the list. See *Customize training loop*.

- `Desc` / `Asc`: select order of project list.
- `Edit`: edit the project name.
- `Delete`: delete the project from list.

6.4 Project: Show training chart and jobs



Show training logs and experimental conditions.

- Select X-axis value by `xAxis` pane.
  - `epoch`, `iteration`, `episode`, `step` and `elapsed_time` are assumed as x-axis.
  - Drop-down list shows only keys existed in `log` files.
- Select values by `yAxis` pane.
  - Left checkboxes are visibility of left axis, right ones are right axis.
  - Line color is selected automatically. To change color, click a job name or a key name, see *Edit a line*.
- Reset setting button
  - Along with axis settings and selected checkboxes, log keys like `main/loss` are also cached on browser storage. The reset button restores cached key, too.
- Save log chart
  - `PNG`: Save log chart as PNG

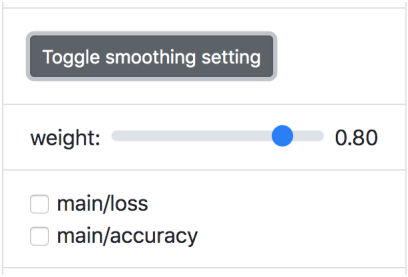
- `Code`: Download Python script. Run the downloaded script then get a chart image using Matplotlib. Lines plotted or not are followed by configuration on Web UI. The script has all log data as JSON.

6.4.1 Highlighting

Fig. 1: This animation is captured on **v0.7.0**

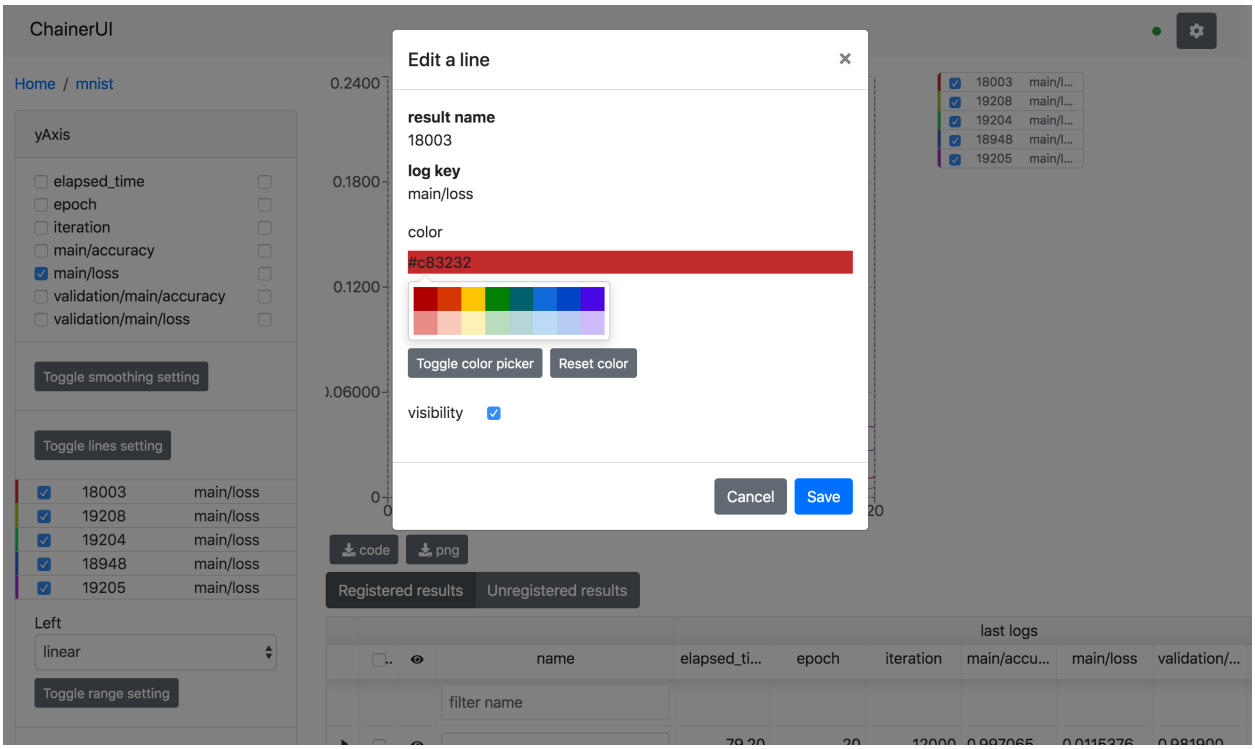
Result table and a log chart are linked each other. A selected result is highlighting for emphasis.

6.4.2 Smoothing



Add smoothing line to help desplaying the overall of trend. Exponential smoothing is used.

6.4.3 Edit a line



Show detail information about the line, and enable to change the line color. To show this modal, click `Toggle lines setting` > a job name or a key name on `yAxis`.

6.4.4 Training job table

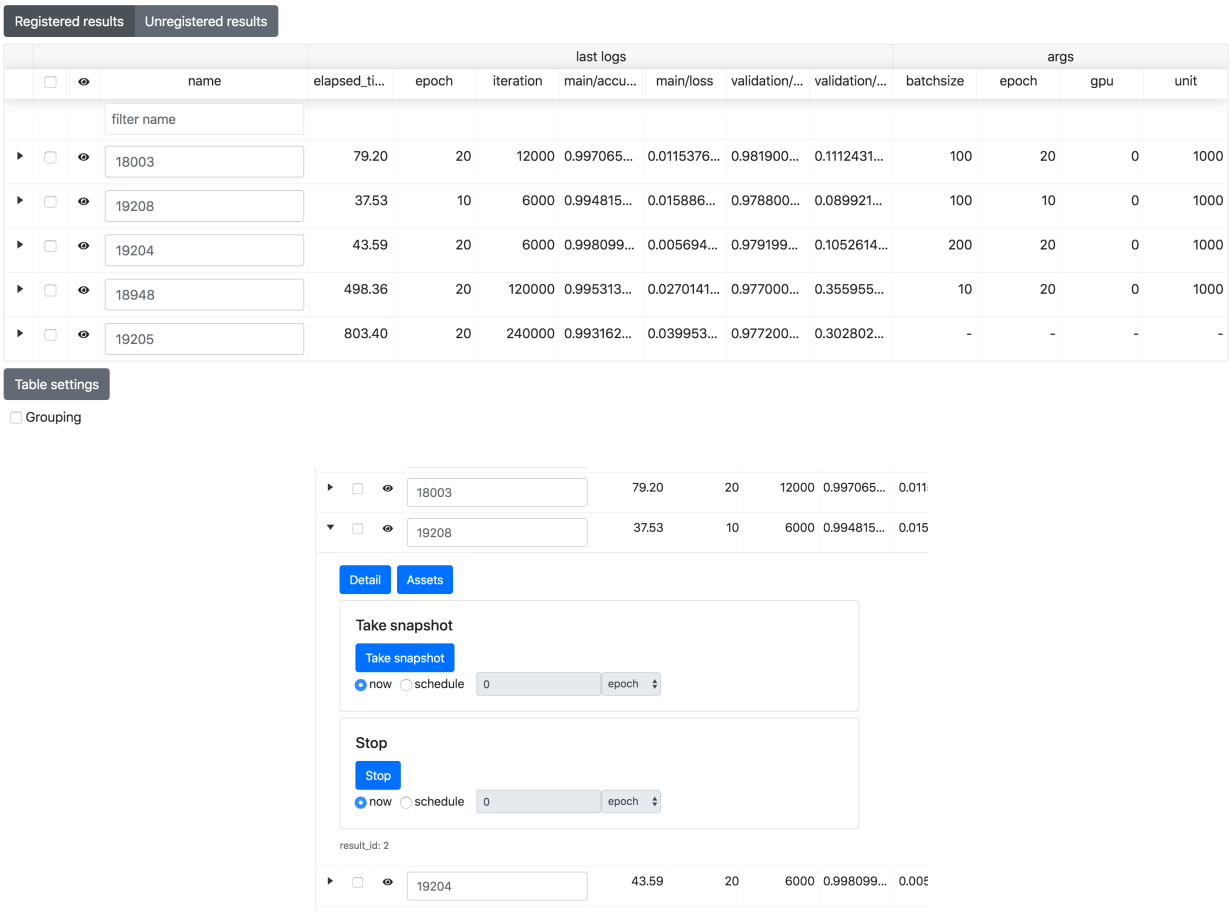
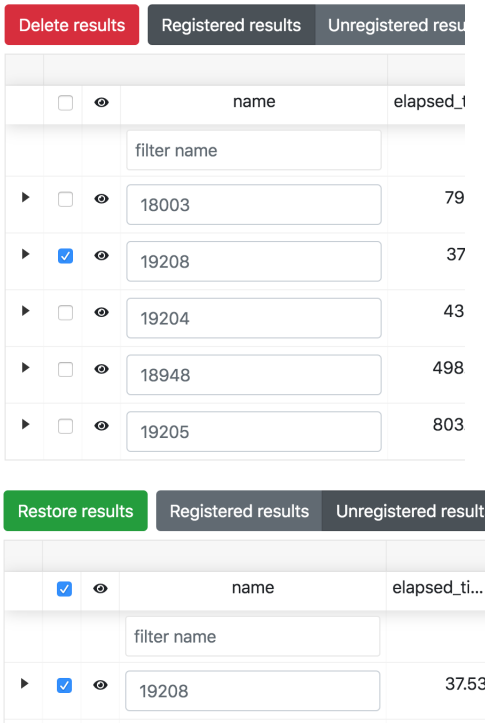


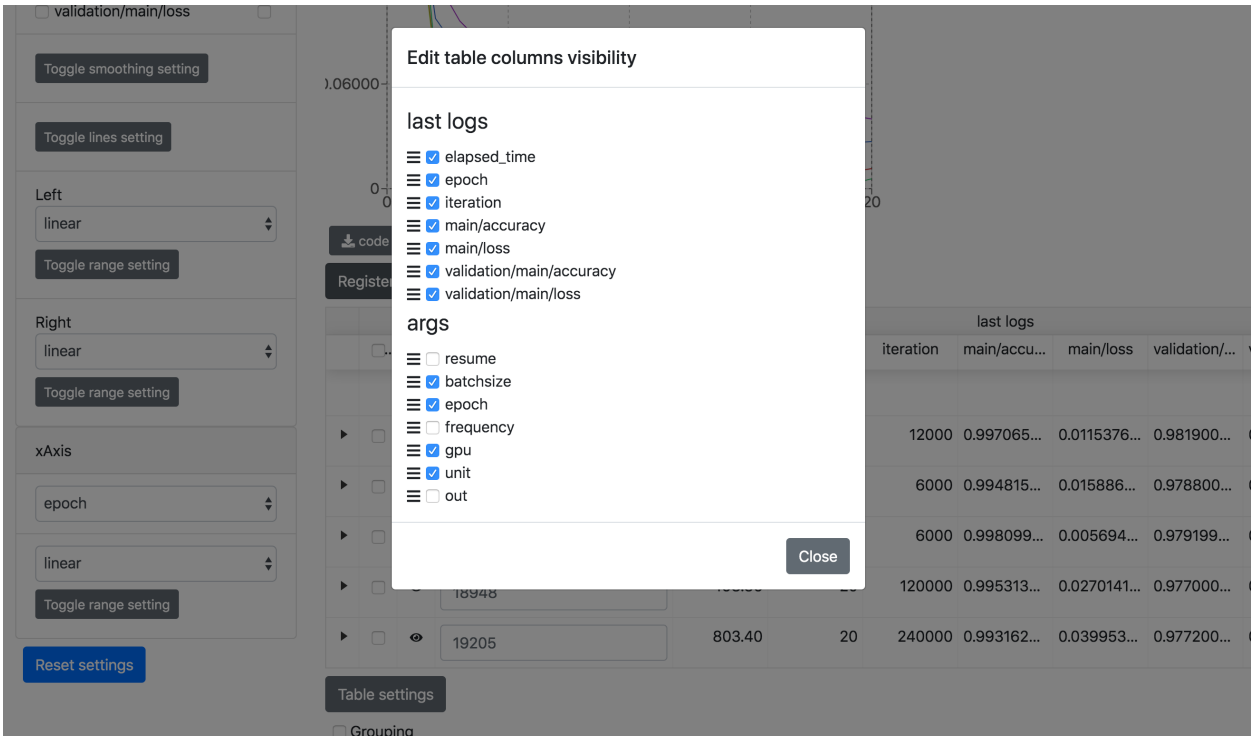
Fig. 2: expanded the second row to show sub components.

The training job table shows brief log information and experimental conditions. Job names are set to the directory name by default. The name can be edit directly on the table. To unregister a result, click `Unregister` button in the expanded row. Expanded row has some operation buttons. These buttons operate similarly to buttons in *Commands pane*.

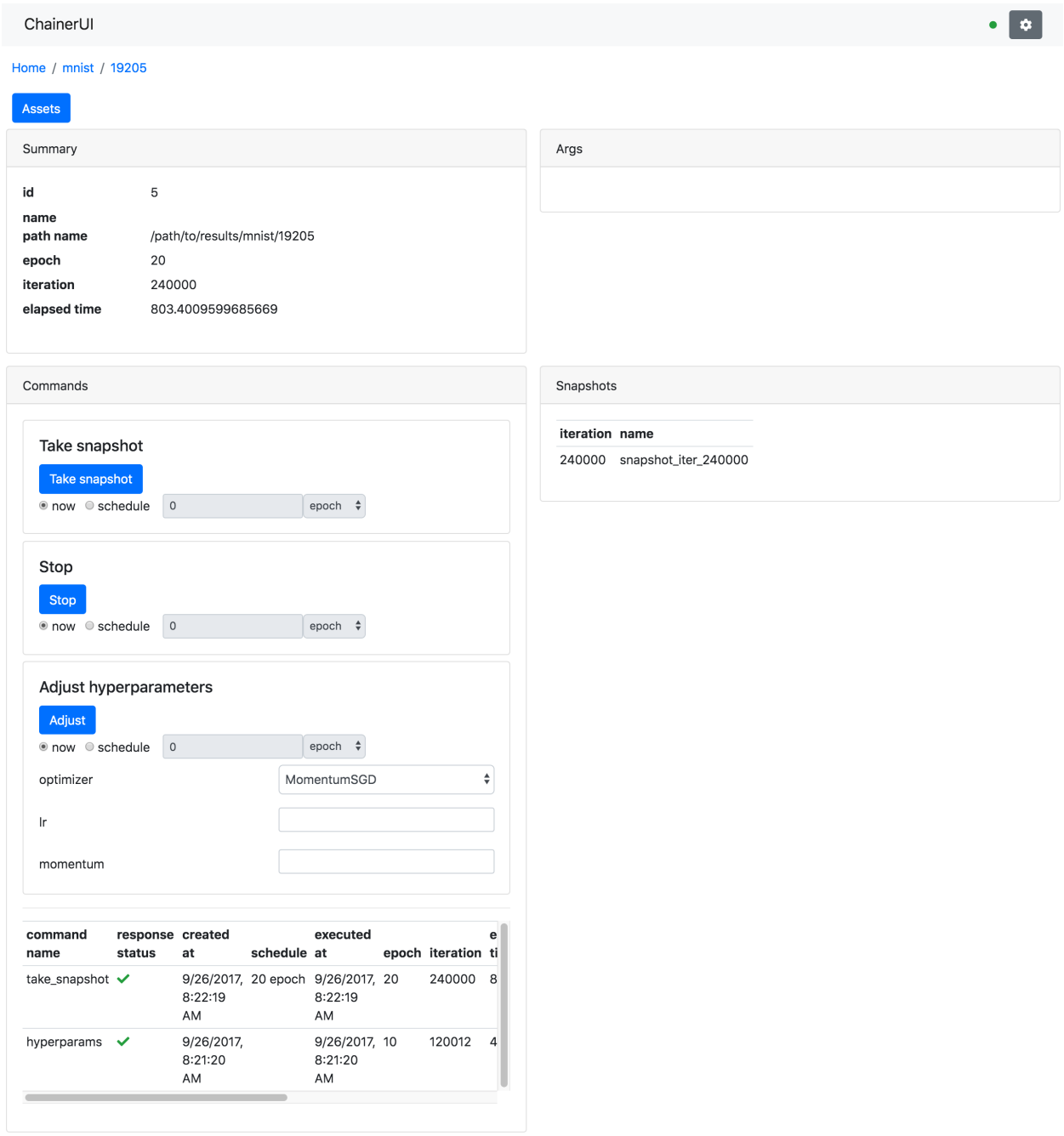
- `Registered results` / `Unregistered results` : These buttons behavior as tab. When need to show unregistered results, select `Unregistered result` tab to show them.
- `Delete results`: When remove results from the result table, check and click `Delete result` button. Deleted resutls are showed on `Unregistered results` tab.
- `Restore results`: When restore deleted result, check the target results on `Unregistered results` tab and click `Restore results` button. Restored results are showed again on `Registered results` tab.



- `filter name`: Filter results by text.
- `Grouping`: Group results by grandparent directory.
- `Table Settings`: Customize visibility and order of table columns.



6.5 Result: Show detailed information of the results



Show detailed information of the training job and support operation of the training loop.

6.5.1 Commands pane

Operation buttons in `Commands` pane allow users to operate the training job. To enable these buttons, the trining job is required to set *CommandsExtension* and click them **within running the job**. For more detail of how to set the extension, see *Operate training loop*.

Take snapshot

Save a training model to the file in NPZ format with using save_napz By default, `snapshot_iter_{.updater.iteration}` file is saved to the result path.

Stop

Stop the trining loop.

Adjust

Adjust the hyperparameters of an optimizer. This function supports only MomentumSGD optimizer.

Command history

The command history is shown on the down of the pane.

CHAPTER 7

ChainerUI command manual

7.1 Server

Run the ChainerUI server. To stop, press `Ctrl+C` on the console:

```
$ chainerui server
```

- `--host` or `-H`: (optional) set original host name
- `--port` or `-p`: (optional) set original port number, set `5000` on default
- `--debug` or `-d`: (optional) run server with debug mode

7.2 Database

Create a ChainerUI database. ChainerUI creates `~/.chainerui/db/chainerui.db` by default and the database references the file:

```
$ chainerui db create
```

Setup the schema for ChainerUI. The `upgrade` operation is always necessary when creating a new database or changing the schema on version up:

```
$ chainerui db upgrade
```

Drop **all** records from database. If continuing to use ChainerUI after executing `drop`, the `create` and `upgrade` operations must be executed.:

```
$ chainerui db drop
```

Warning: When removing selected projects, don't use the `drop` commands. Use `Delete` button on *project list page*.

7.3 Project

ChainerUI manages multiple projects and each project manages multiple training logs. Once a project directory is created, ChainerUI starts to monitor the directory and register log files under the directory. The searching process is run recursively and nested directories are available:

```
$ chainerui project create -d PROJECT_DIR
```

- `-d`: (required) target path
- `-n`: (optional) name of project. use directory name on default.

7.4 Common option

7.4.1 `--db`

When use external database, set `--db` option to use it. For example, when use SQLite with an original database file placed at `/path/to/original.db`, initialize commands are:

```
$ chaiherui --db sqlite:////path/to/original.db db upgrade
$ chaiherui --db sqlite:////path/to/original.db server
```

This `--db` option is given priority over environment variable `CHAINERUI_DB_URL`. More detail, see *Use external database*

CHAPTER 8

Module Reference

8.1 Trainer extensions

class chainerui.extensions.**CommandsExtension**(*trigger=(1, 'iteration')*, *receivers={}*, *file_name='commands'*)

Trainer extension to enable command operation by output file

This extension monitors a file for command created on *trainer.out* path, and execute each command when append the file.

8.2 Asset summaries

chainerui.summary.**set_out**(*path*)

Set output path.

Summary module requires output directory. Once set output path using this function, summary module shares the path.

Parameters **path** (*str*) – directory path of output.

chainerui.summary.**image**(*images*, *name=None*, *ch_axis=1*, *row=0*, *mode=None*, *batched=True*, *out=None*, *subdir=''*, *timeout=5*, ***kwargs*)

Summarize images to visualize.

Array of images are converted as image format (PNG format on default), saved to output directory, and reported to the ChainerUI server. The images are saved every called this function. The images will be shown on *assets* endpoint vertically. If need to aggregate images in a row, use *reporter()*.

Examples of how to set arguments:

```
>>> from chainerui import summary
>>> summary.set_out('path/to/log')  # same as 'log' file directory
>>>
>>> x = np.zeros((10, 3, 5, 5))  # = [Batchsize, Channel, Height, Width]
```

(continues on next page)

(continued from previous page)

```
>>> summary.image(x, name='test')  # images are tiled as 1x10
>>> summary.image(x, name='test', row=2)  # images are tiled as 2x5
>>>
>>> x = np.zeros((3, 5, 5))  # = [C, H, W]
>>> # need to set as a non-batched image and channel axis explicitly
>>> summary.image(x, name='test', ch_axis=0, batched=False)
>>>
>>> x = np.zeros((10, 5, 5, 3))  # = [B, H, W, C]
>>> # need to set channel axis explicitly
>>> summary.image(x, name='test', ch_axis=-1, row=2)
>>>
>>> x = np.zeros((5, 5, 3))  # = [H, W, C]
>>> # need to set as a non-batched image
>>> summary.image(x, name='test', ch_axis=-1, batched=False)
>>>
>>> x = np.zeros((10, 5, 5))  # = [B, H, W], grayscale images
>>> summary.image(x, name='test')  # image are tiled as 1x10
>>> summary.image(x, name='test', row=2)  # image are tiled as 2x5
>>>
>>> x = np.zeros((5, 5)) # = [H, W], a grayscale image
>>> # need to set as a non-bathed image
>>> summary.image(x, name='test', batched=False)
```

Add description about the image:

```
>>> summary.image(x, name='test', epoch=1, iteration=100)
>>> # 'epoch' and 'iteration' column will be shown.
```

Parameters

- **images** (`numpy.ndarray` or `cupy.ndarray` or `chainer.Variable`) – batch of images. If Number of dimension is 3 (or 2 when set *batched=False*), the pixels assume as black and white image.
- **name** (*str*) – name of image. set as column name. when not setting, assigned `'image'`.
- **ch_axis** (*int*) – index number of channel dimension. set 1 by default. if the images don't have channel axis, this parameter is ignored.
- **row** (*int*) – row size to visualize batched images. when set 0, show on unstuck. if images set only one image, the row size will be ignored.
- **mode** (*str*) – if the images are not RGB or RGBA space, set their color space code. ChainerUI supports 'HSV'.
- **batched** (*bool*) – if the image is not batched, set `False`.
- **out** (*str*) – directory path of output.
- **subdir** (*str*) – sub-directory path of output.
- ****kwargs** (*dict*) – key-value pair to show as description. regardless of empty or not, timestamp on created the image is added.

`chainerui.summary.`**audio**(*audio*, *sample_rate*, *name=None*, *out=None*, *subdir=''*, *timeout=5*, ***kwargs*)

Summarize audio files to listen on a browser.

An sampled array is converted as WAV audio file, saved to output directory, and reported to the ChainerUI server. The audio file is saved every called this function. The audio file will be listened on *assets* endpoint vertically. If need to aggregate audio files in row, use *reporter()*.

Example of how to set arguments:

```
>>> from chainerui import summary
>>> summary.set_out('path/to/output')
>>> rate = 44100
>>>
>>> sampled_array = np.random.uniform(-1, 1, 16000)
>>> summary.audio(sampled_array, rate, name='test')
>>> # sampled_array can be listened on a browser.
```

Add description about the audio file:

```
>>> summary.audio(
...     sampled_array, rate, name='test', epoch=1, iteration=100)
>>> # 'epoch' and 'iteration' column will be shown.
```

Parameters

- **audio** (numpy.ndarray or cupy.ndarray or chainer.Variable) – sampled wave array.
- **sample_rate** (*int*) – sampling rate.
- **name** (*str*) – name of image. set as column name. when not setting, assigned 'audio'.
- **out** (*str*) – directory path of output.
- **subdir** (*str*) – sub-directory path of output.
- ****kwargs** (*dict*) – key-value pair to show as description. regardless of empty or not, timestamp on created the image is added.

chainerui.summary.**text**(*text*, *name=None*, *out=None*, *timeout=5*, ***kwargs*)

Summarize texts to show on a browser.

Texts generated by training model is saved as asset and reported to the ChainerUI.

Parameters

- **text** (*str*) – generated text.
- **name** (*str*) – name of text. set as column name. when not setting, assigned 'text'.
- **out** (*str*) – directory path of output.
- ****kwargs** (*dict*) – key-value pair to show as description. regardless of empty or not, timestamp on created the text is added.

chainerui.summary.**reporter**(*prefix=None*, *out=None*, *subdir=''*, *timeout=5*, ***kwargs*)

Summarize media assets to visualize.

reporter function collects media assets by the with statement and aggregates in same row to visualize. This function returns an object which provides the following methods.

- *image()*: collect images. almost same as *image()*
- *audio()*: collect audio. almost same as *audio()*
- *text()*: collect text. almost same as *text()*

Example of how to set several assets:

```
>>> from chainerui import summary
>>> summary.set_out('path/to/output')  # same as 'log' file directory
>>>
>>> image_array1 = np.zeros((1, 3, 224, 224))
>>> image_array2 = np.zeros((1, 3, 224, 224))
>>> audio_array = np.random.uniform(-1, 1, 16000)
>>>
>>> from chainerui.summary import reporter
>>> with reporter(epoch=1, iteration=10) as r:
...     r.image(image_array1)
...     r.image(image_array2)
...     r.audio(audio_array, 44100)
>>> # image_array1 and image_array2 are visualized on a browser
>>> # audio_array can be listened on a browser
```

Parameters

- **prefix** (*str*) – prefix of column name.
- **out** (*str*) – directory path of output.
- **subdir** (*str*) – sub-directory path of output.
- ****kwargs** (*dict*) – key-value pair to show as description. regardless of empty or not, timestamp is added.

_Reporter.**image**(*images*, *name=None*, *ch_axis=1*, *row=0*, *mode=None*, *batched=True*, *subdir=''*)

Summarize images to visualize.

Parameters

- **images** (numpy.ndarray or cupy.ndarray or chainer.Variable) – batch of images. If Number of dimension is 3 (or 2 when set *batched=False*), the pixels assume as black and white image.
- **name** (*str*) – name of image. set as column name. when not setting, assigned 'image' + sequential number.
- **ch_axis** (*int*) – index number of channel dimension. set 1 by default. if the images don't have channel axis, this parameter is ignored.
- **row** (*int*) – row size to visualize batched images. when set 0, show on unstuck. if images set only one image, the row size will be ignored.
- **mode** (*str*) – if the images are not RGB or RGBA space, set their color space code. ChainerUI supports 'HSV'.
- **batched** (*bool*) – if the image is not batched, set False.
- **subdir** (*str*) – sub-directory path of output.

_Reporter.**audio**(*audio*, *sample_rate*, *name=None*, *subdir=''*)

Summarize audio to listen on web browser.

Parameters

- **audio** (numpy.ndarray or cupy.ndarray or chainer.Variable) – sampled wave array.
- **sample_rate** (*int*) – sampling rate.

- **name** (*str*) – name of image. set as column name. when not setting, assigned `'audio'` + sequential number.
- **subdir** (*str*) – sub-directory path of output.

`_Reporter.text`(*text*, *name=None*)

Summarize texts to show on a browser.

Parameters

- **text** (*str*) – generated text.
- **name** (*str*) – name of text. set as column name. when not setting, assigned `'text'`.

8.3 Web client

`chainerui.init`(*url=None*, *project_name=None*, *result_name=None*, *overwrite_result=False*, *crawlable=False*, *conditions=None*)

Initialize client tools

Initialize client object, then setup project and result. Even if some errors are occurred, client object is set `None` and without exception.

Parameters

- **url** (*str*) – ChainerUI server URL. set `'localhost:5000'` on default.
- **project_name** (*str*) – project name is set from project path, working directory on default. If set, ChainerUI shows the name instead the project path.
- **result_name** (*str*) – result name is set project path + start time on default. If set, ChainerUI shows the name + start time instead the path.
- **overwrite_result** (*bool*) – the client tool make different job results every time on default. If set `True`, the client tool posts logs on the same result.
- **crawlable** (*bool*) – to inform server not to crawl physical logs.
- **conditions** (`argparse.Namespace` or dict) – Experiment conditions to show on a job table. Keys are show as table header and values are show at a job row.

`chainerui.log_reporter`()

Log reporter via POST API

Return a callback function to post a log to a ChainerUI server. If the initialization (see `chainerui.init()`) is failed, the callback function do nothing when called. If the initialization is done but fail to send the log by some kind of error, the log is cached and try to post it next time with new one.

The callback function is expected to use with `postprocess` option of Chainer `LogReport` extension:

```
>>> chainerui.init()
>>>
>>> trainer = chainer.training.Trainer(...)
>>> trainer.extend(
>>>     extensions.LogReport(postprocess=chainerui.log_reporter()))
```

Returns function.

Return type func

chainerui.**log**(*value*)
Send log

Send log data and will be shown ad training log on web browser.

Example:

```
>>> chainerui.init()
>>> # ...
>>> stats = {
>>>     'epoch': epoch,
>>>     'train/loss': loss,
>>>     }
>>> chainerui.log(stats)
```

Parameters **value** (*dict*) – target log.

8.4 Utilities

class chainerui.utils.**LogReport**(*out_path*, *conditions=None*)
Util class to output 'log' file.

This class supports to output 'log' file. The file spec follows `chainer.extensions.LogReport`, however, 'epoch' and 'iteration' are not set automatically, and need to set these values.

Parameters

- **out_path** (*str*) – Output directory name to save conditions.
- **conditions** (`argparse.Namespace` or dict) – Experiment conditions to show on a job table. Keys are show as table header and values are show at a job row.

chainerui.utils.**save_args**(*conditions*, *out_path*)
A util function to save experiment condition for job table.

Parameters

- **conditions** (`argparse.Namespace` or dict) – Experiment conditions to show on a job table. Keys are show as table header and values are show at a job row.
- **out_path** (*str*) – Output directory name to save conditions.

8.5 External library support

class chainerui.contrib.ignite.handler.**OutputHandler**(*tag*, *metric_names=None*, *output_transform=None*, *another_engine=None*, *global_step_transform=None*, *interval_step=-1*)
Handler for ChainerUI logger

A helper for handler to log engine's output, specialized for ChainerUI. This handler sets 'epoch', 'iteration' and 'elapsed_time' automatically, these are default x axis to show.

```
from chainerui.contrib.ignite.handler import OutputHandler
train_handler = OutputHandler(
    'train', output_transform=lambda o: {'param': o})
val_handler = OutputHandler('val', metric_names='all')
```

Parameters

- **tag** (*str*) – use for a prefix of parameter name, will show as {tag}/{param}
- **metric_names** (*str or list*) – keys names of `list` to monitor. set `'all'` to get all metrics monitored by the engine.
- **output_transform** (*func*) – if set, use this function to convert output from `engine.state.output`
- **another_engine** (`ignite.engine.Engine`) – if set, use for getting global step. This option is deprecated from 0.3.
- **global_step_transform** (*func*) – if set, use this to get global step.
- **interval_step** (*int*) – interval step for posting metrics to ChainerUI server.

class chainerui.contrib.ignite.handler.**ChainerUILogger**

Logger handler for ChainerUI

A helper logger to post metrics to ChainerUI server. Attached handlers are expected using `chainerui.contrib.ignite.handler.OutputHandler`. A tag name of handler must be unique when attach several handlers.

```
from chainerui.contrib.ignite.handler import OutputHandler
train_handler = OutputHandler(...)
val_handler = OutputHandler(...)

from ignite.engine.engine import Engine
train_engine = Engine(...)
eval_engine = Engine(...)

from chainerui.contrib.ignite.handler import ChainerUILogger
logger = ChainerUILogger()
logger.attach(
    train_engine, log_handler=train_handler,
    event_name=Events.EPOCH_COMPLETED)
logger.attach(
    eval_engine, log_handler=val_handler,
    event_name=Event.EPOCH_COMPLETED)
```

CHAPTER 9

Indices and tables

- genindex
- search

Index