chainerui Documentation

Release 0.7.0

Preferred Networks, inc.

Nov 26, 2019

ChainerUI Documents

CHAPTER 1

Installation Guide

1.1 Dependencies

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

Listing 1: requirements.txt

```
enum34>=1.1.6; python_version < '3.4'
msgpack>=0.5.6
Flask>=1.0.2
sqlalchemy>=1.1.18
alembic>=1.0.0
chainer>=3.0.0
gevent>=1.2.2
structlog>=18.2.0
```

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 your first project
$ chainerui project create -d examples -n example-project
```

Run ChainerUI server:

```
$ chainerui server
```

Open http://localhost:5000/ and select "example-project", then show 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:v0.4.0
$ docker run -d -p 5000:5000 -v $PWD:/projects --name chainerui chainer/chainerui:v0.
↪4.0
$ # then ChainerUI server is running
$ # create project via HTTP
$ curl http://localhost:5000/api/v1/projects -X POST -H "Content-Type: application/
↪json" -d '{"project":{"name":"example-project","path_name":"/projects/examples"}}'
```

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

Form more detailed usage, see *Use Docker*.

1.5 Browser compatibility

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

- Firefox
- Chrome

CHAPTER 2

Getting started

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

(continues on next page)

(continued from previous page)

```
|--- 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*:

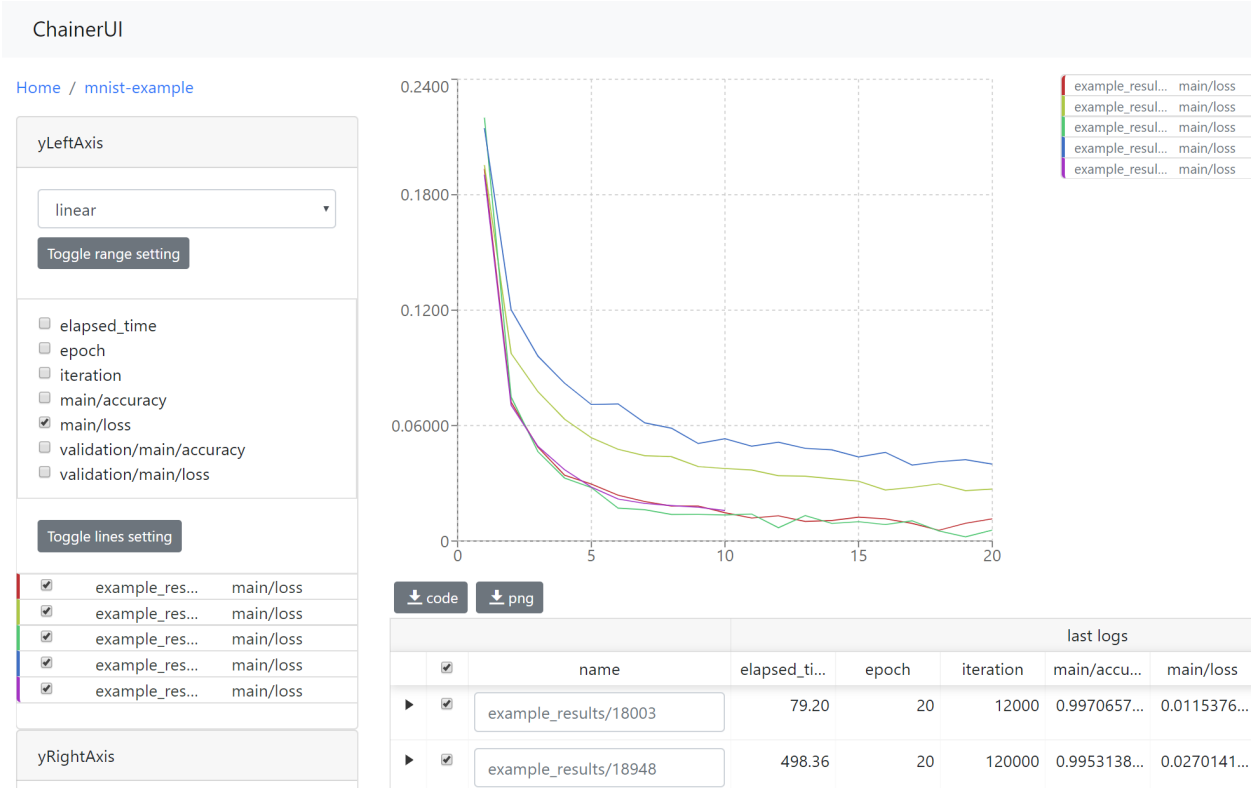
2.4 Customize training loop

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

Note: examples/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/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
```

(continues on next page)

(continued from previous page)

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

2.4.2 Experimental conditions

			last logs			args						
		name	epoch	iteration	elapsed_ti...	resume	batchsize	epoch	frequency	gpu	unit	out
▸	☑	example_results/18003	20	12000	79.20		100	20	-1	0	1000	results
▸	☑	example_results/18948	20	120000	498.36		10	20	-1	0	1000	results
▸	☑	example_results/19204	20	6000	43.59		200	20	-1	0	1000	results
▸	☑	example_results/19205	20	240000	803.40	-	-	-	-	-	-	-
▸	☑	example_results/19208	10	6000	37.53		100	10	-1	0	1000	results

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, or in *result page*, click `Detail` button in expanded row.

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

Setup example of a brief extract MNIST example:

Commands

Take snapshot

Take snapshot

now schedule 0 epoch

Stop

Stop

now schedule 0 epoch

Adjust hyperparameters

Adjust

now schedule 0 epoch

optimizer MomentumSGD

lr

momentum

command name	response status	created at	schedule	executed at	epoch	iteration	elapsed time	request body	response body
take_snapshot	✔	2017/9/26 16:44:33	4 epoch	2017/9/26 16:44:35	4	2400	76.97		

Fig. 2: commands pane of result page

```
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())
    # [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 Docker

ChainerUI provides `Dockerfile` from version 0.4.0 and ChainerUI server can be run on a Docker container.

3.1 Get Docker container

The Docker container can be got from DockerHub or built yourself. When getting the container from DockerHub, set the latest version to the tag. The below code gets the version 0.4.0 container:

```
$ docker pull chainer/chainerui:v0.4.0
```

When building Docker container yourself, use `Dockerfile` placed in docker directory:

```
$ git clone https://github.com/chainer/chainerui.git
$ cd chainerui
$ docker build -t chainer/chainerui:v0.4.0 -f docker/Dockerfile .
```

3.2 Run ChainerUI server

The Docker container has already setup a command to start the server, and requires port number to be linked to host (`-p` option) and volume to be mounted (`-v` option):

```
$ docker run -d -p 5000:5000 -v /path/to/job:/projects --name chainerui chainer/
↪chainerui:v0.4.0
```

- `-p 5000:5000`: the container exposes port `5000` for ChainerUI server.
- `-v /path/to/job:/projects`: the container setups `/projects` as Docker volumes. Remember that Docker volume does not support symbolic link and relative path.

ChainerUI server will run, open http://localhost:5000/ .

When stop the container:

```
$ docker stop chainerui
```

When restart the container:

```
$ docker start chainerui
```

Warning: ChainerUI stores all data, such as logs, args and so on, to the own DB created in the image. **These data are removed when the container is removed.**

3.3 Create a project

To store data such as logs and show a log chart, a project with a result directory path is needed. There are 2 ways to register projects to the server, via HTTP or via `docker exec`. For more detail about project function, see *Create a project*

Note: The project's path is viewed from the container: guest OS, **not viewed from the host OS**. For example, the result directory is below structure and the container is mounted as `-v /path/to/job:/projects`:

```
On host OS
/path/to/job
  |--- results
         |--- result1
                |--- log
         |--- result2
                |--- log
```

`/path/to/job` is mounted to `/projects` in guest OS, so the project's path is `/projects/results`, viewed from guest OS.

3.3.1 Via HTTP

POST a project information to the endpoint `/projects`, following is an example command using `curl`:

```
$ curl http://localhost:5000/api/v1/projects -X POST -H "Content-Type: application/
↪json" -d '{"project":{"name":"PROJECT_NAME","path_name":"/projects/results"}}'
```

3.3.2 Call command directly

ChainerUI command is enabled in the container:

```
$ docker exec -it chainerui /bin/bash
# chainerui project create -d /projects/result -n PROJECT_NAME
```

CHAPTER 4

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.

4.1 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
```

4.2 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
```

4.3 Example: MySQL

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

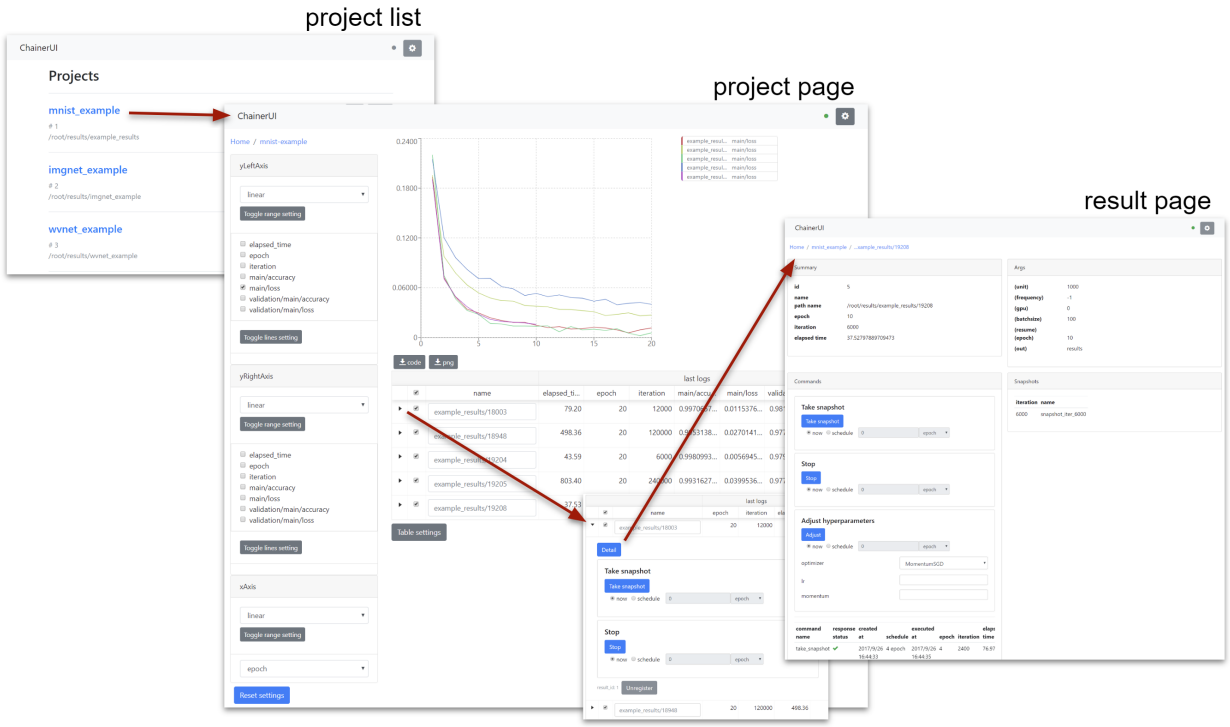
```
$ docker pull mysql:8.0.12
$ docker run --name mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root_pass -e MYSQL_
↪USER=user -e MYSQL_PASSWORD=pass -e MYSQL_DATABASE=chainerui -d mysql:8.0.12

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

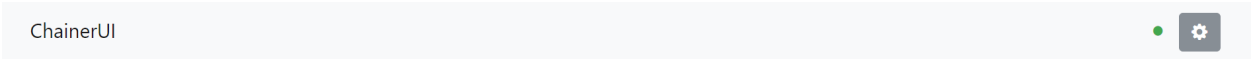
CHAPTER 5

User interface manual

Page transition flow:

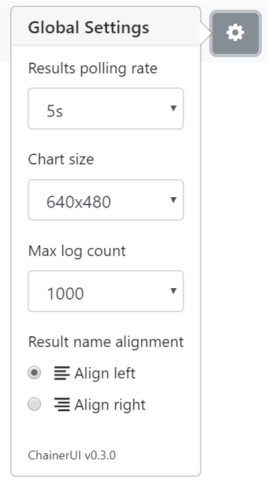


5.1 Header



-  : 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

5.2 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.

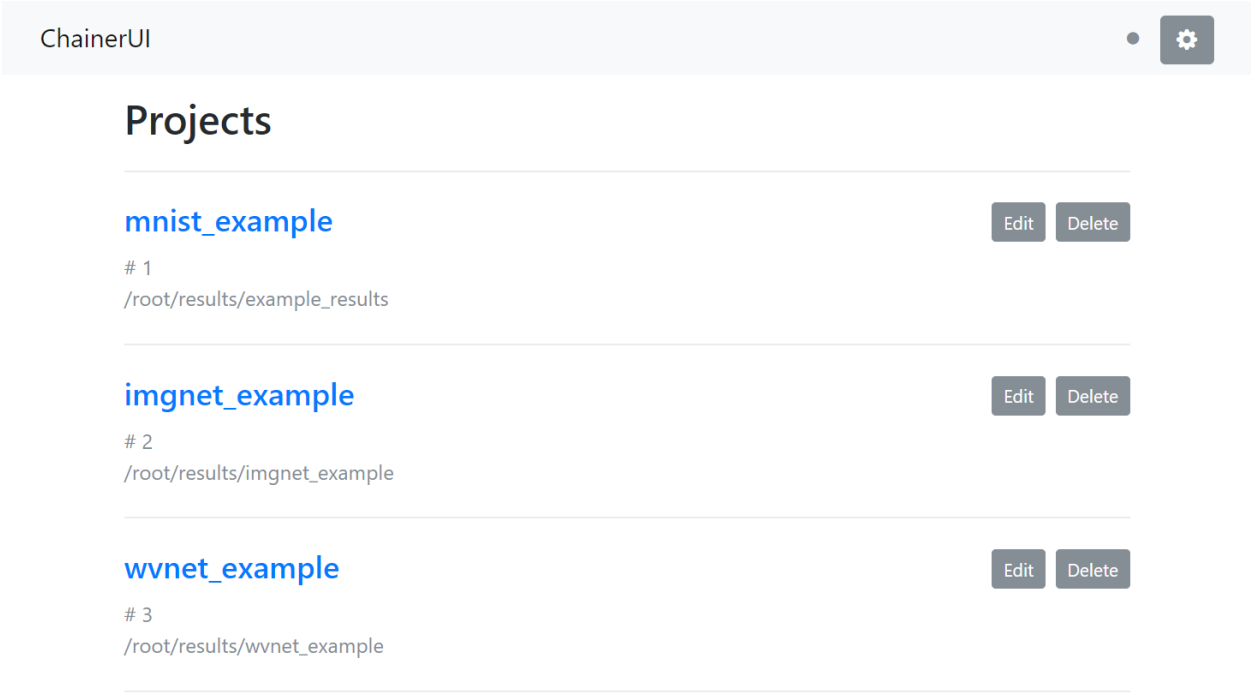
Max log count

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.

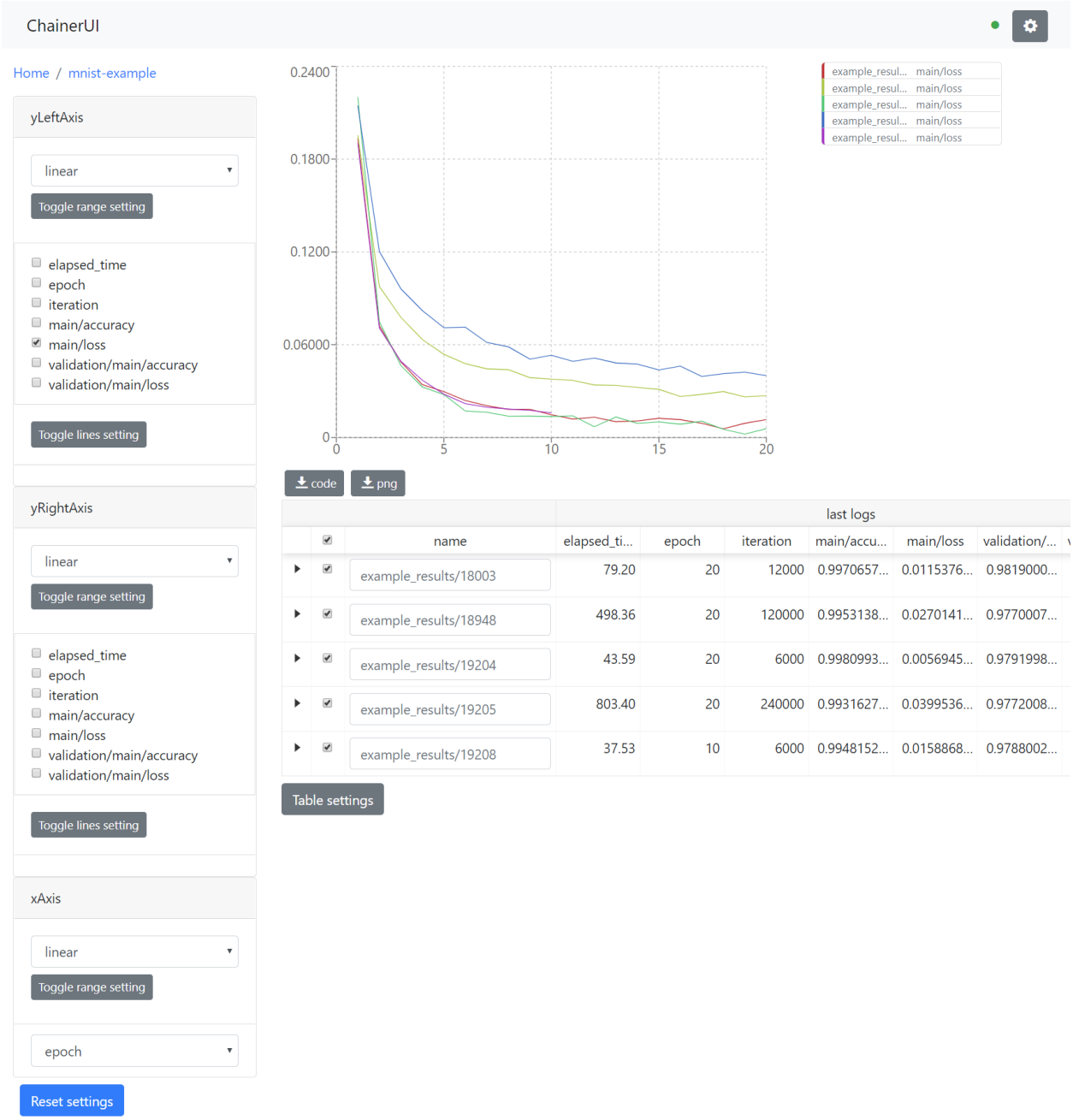
5.3 Home: Project list



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*.

- `Edit`: edit the project name.
- `Delete`: delete the project from list.

5.4 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 `yLeftAxis` and `yRightAxis` panes.
  - Line color is selected automatically. To change color, click a job name or a key name, see *Edit a line*.

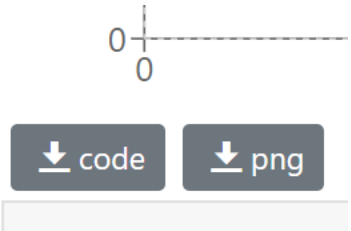
- Select training job to show on a chart.
- Reset setting button
  - Along with axis settings and selected checkboxes, log keys like `main/loss` are also cached on browser storage. The reset button restores cached key, too.

5.4.1 Highlighting

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

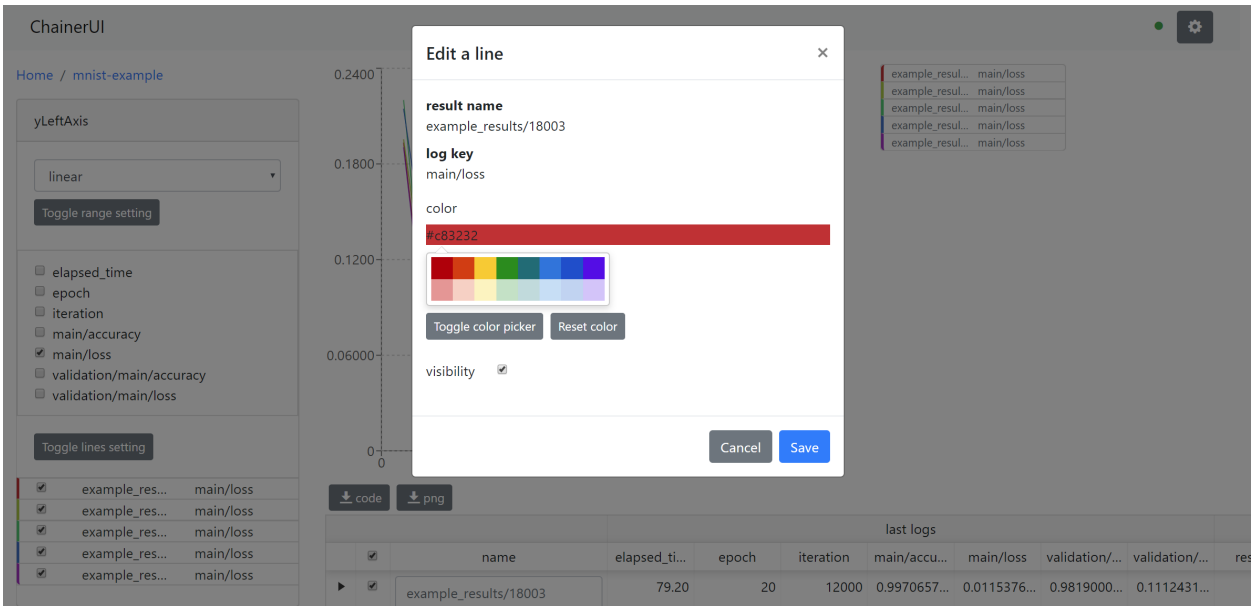
Result table and a log chart are linked each other. A selected result is highlighting for emphasis.

5.4.2 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.

5.4.3 Edit a line



Show detail information about the line, and enable to change the line color. To show this modal, click a job name or a key name on `yLeftAxis` (`yRightAxis`).

5.4.4 Training job table

			last logs				args					
		name	epoch	iteration	elapsed_ti...	resume	batchsize	epoch	frequency	gpu	unit	out
▸	☑	example_results/18003	20	12000	79.20		100	20	-1	0	1000	results
▸	☑	example_results/18948	20	120000	498.36		10	20	-1	0	1000	results
▸	☑	example_results/19204	20	6000	43.59		200	20	-1	0	1000	results
▸	☑	example_results/19205	20	240000	803.40	-	-	-	-	-	-	-
▸	☑	example_results/19208	10	6000	37.53		100	10	-1	0	1000	results

			last logs				args					
		name	epoch	iteration	elapsed_ti...	resume	batchsize	epoch	frequency	gpu	unit	out
▾	☑	example_results/18003	20	12000	79.20		100	20	-1	0	1000	results

Detail

Take snapshot

Take snapshot

now schedule 0 epoch

Stop

Stop

now schedule 0 epoch

result_id: 1 Unregister

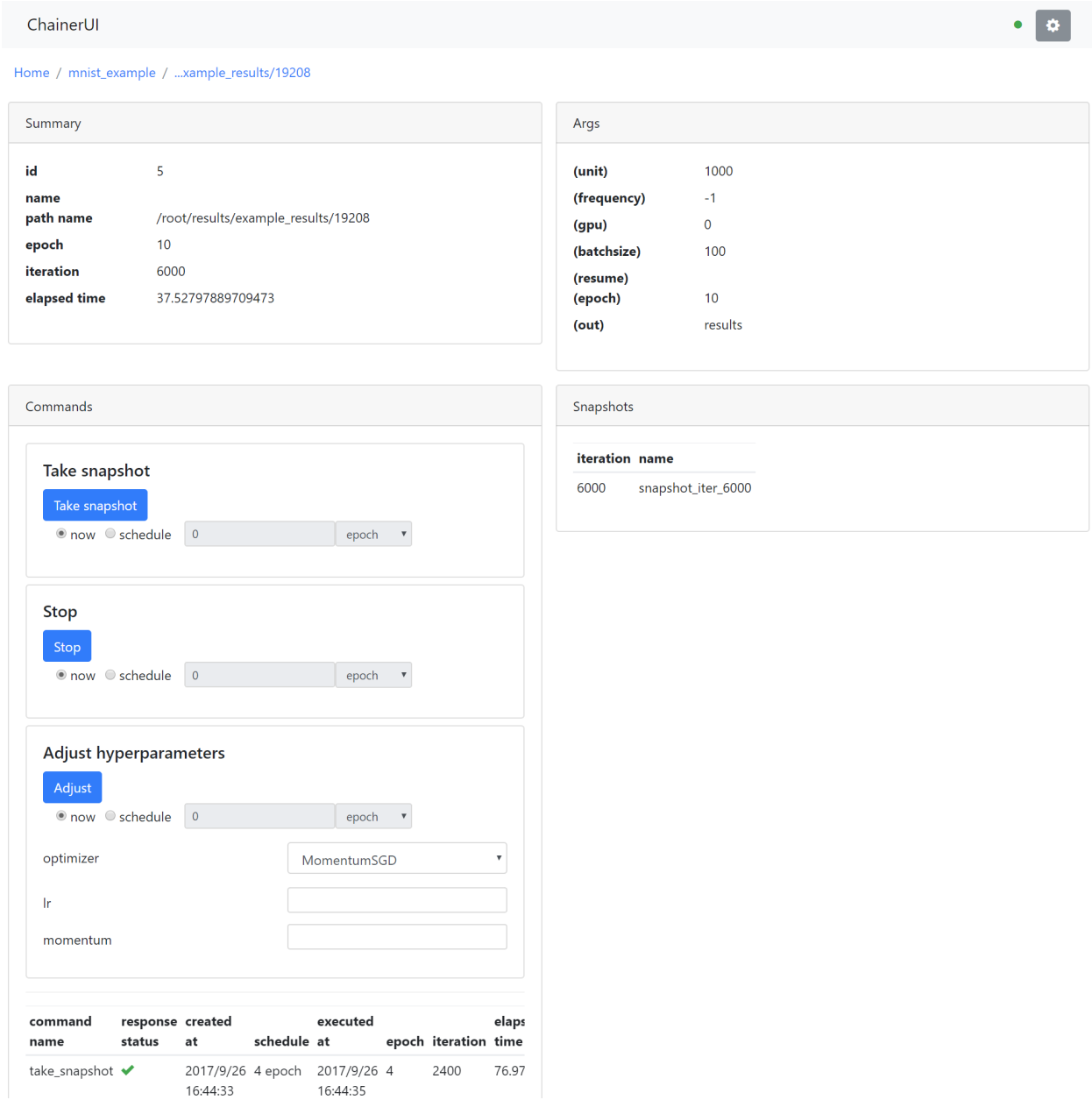
▸	☑	example_results/18948	20	120000	498.36		10	20	-1	0	1000	results

Fig. 2: expanded the first 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*.

Note: [Known problem] Once a result is unregistered, a result with the same name cannot be restored on the result table. This will be fixed in future.

5.5 Result: Show detailed information of the results



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

5.5.1 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 6

ChainerUI command manual

6.1 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

6.2 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*.

6.3 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.

6.4 Common option

6.4.1 `--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 7

Module Reference

7.1 chainerui.extensions

7.2 chainerui.utils

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.

CHAPTER 8

Indices and tables

- genindex
- search

Index

L

S