# Development

Please follow our established coding style including variable names, module imports, and function definitions.

The Pyro codebase follows the [PEP8 style guide](https://www.python.org/dev/peps/pep-0008/)

(which you can check with `make lint`) and follows

[`isort`](https://github.com/timothycrosley/isort) import order (which you can enforce with `make format`).

# Setup

First install [PyTorch](http://pytorch.org/).

Then, install all the dev dependencies for Pyro.

```sh

make install

```

or explicitly

```sh

pip install -e .[dev]

```

# Testing

Before submitting a pull request, please autoformat code and ensure that unit tests pass locally

```sh

make format # runs isort

make test # linting and unit tests

```

If you've modified core pyro code, examples, or tutorials, you can run more comprehensive tests locally (after first adding any new files to the appropriate `tests/` script)

```sh

make test-examples # test examples/

make integration-test # longer-running tests (may take hours)

make test-cuda # runs unit tests in cuda mode

```

To run all tests locally in parallel, use the `pytest-xdist` package

```sh

pip install pytest-xdist

pytest -vs -n auto

```

To run a single test from the command line

```sh

pytest -vs {path\_to\_test}::{test\_name}

# or in cuda mode

CUDA\_TEST=1 PYRO\_TENSOR\_TYPE=torch.cuda.DoubleTensor pytest -vs {path\_to\_test}::{test\_name}

```

## Testing Tutorials

We run some tutorials on travis to avoid bit rot.

Before submitting a new tutorial, please run `make scrub` from

the top-level pyro directory in order to scrub the metadata in

the notebooks.

To enable a tutorial for testing

1. Add a line `smoke\_test = ('CI' in os.environ)` to your tutorial. Our test

scripts only test tutorials that contain the string `smoke\_test`.

2. Each time you do something expensive for many iterations, set the number

of iterations like this:

```py

for epoch in range(200 if not smoke\_test else 1):

...

```

You can test locally by running `make test-tutorials`.

# Profiling

The profiler module contains scripts to support profiling different

Pyro modules, as well as test for performance regression.

To run the profiling utilities, ensure that all dependencies for profiling are satisfied,

by running `make install`, or more specifically, `pip install -e .[profile]`.

There are some generic test cases available in the `profiler` module. Currently, this supports

only the `distributions` library, but we will be adding test cases for inference methods

soon.

#### Some useful invocations

To get help on the parameters that the profiling script takes, run:

```sh

python -m profiler.distributions --help

```

To run the profiler on all the distributions, simply run:

```sh

python -m profiler.distributions

```

To run the profiler on a few distributions by varying the batch size, run:

```sh

python -m profiler.distributions --dist bernoulli normal --batch\_sizes 1000 100000

```

To get more details on the potential sources of slowdown, use the `cProfile` tool

as follows:

```sh

python -m profiler.distributions --dist bernoulli --tool cprofile

```

# Submitting

For larger changes, please open an issue for discussion before submitting a pull request.

For relevant design questions to consider, see past

[design documents](https://github.com/pyro-ppl/pyro/wiki/Design-Docs).

In your PR, please include:

- Changes made

- Links to related issues/PRs

- Tests

- Dependencies

If you add new files, please run `make license` to automatically add copyright headers.

For speculative changes meant for early-stage review, include `[WIP]` in the PR's title.

(One of the maintainers will add the `WIP` tag.)