flow Documentation

Release 0.1

Cathy Wu

Oct 01, 2018

Contents:

Flow is a computational framework for deep RL and control experiments for traffic microsimulation. Visit our website for more information.

Flow is a work in progress - input is welcome. Available documentation is limited for now. Tutorials are available in iPython notebook format.

If you are looking for Akvo Flow, their documentation can be found at http://flowsupport.akvo.org.

CHAPTER 1

Introduction

1.1 Why Flow?

Traffic systems can often be modeled by complex (nonlinear and coupled) dynamical systems for which classical analysis tools struggle to provide the understanding sought by transportation agencies, planners, and control engineers, mostly because of difficulty to provide analytical results on these. Deep reinforcement learning (RL) provides an opportunity to study complex traffic control problems involving interactions of humans, automated vehicles, and sensing infrastructure. The resulting control laws and emergent behaviors of the vehicles provide insight and understanding of the potential for automation of traffic through mixed fleets of autonomous and manned vehicles.

Flow is a traffic control benchmarking framework. Its provides a suite of traffic control scenarios (benchmarks), tools for designing custom traffic scenarios, and integration with deep reinforcement learning and traffic microsimulation libraries.

Table of contents

CHAPTER 2

Local Installation

To get Flow running, you need three things: Flow, SUMO, and (recommended to explore the full suite of Flow's capabilities) a reinforcement learning library (RLlib/rllab). If you choose not to install a reinforcement learning library, you will still be able to build and run SUMO-only traffic tasks, but will not be able to run experiments which require learning agents. Once each component is installed successfully, you might get some missing module bugs from Python. Just install the missing module using your OS-specific package manager / installation tool. Follow the shell commands below to get started.

It is highly recommended that users install Anaconda or Miniconda for Python and the setup instructions will assume that you are doing so.

2.1 a. Installing Flow and SUMO

In this section we install Flow as well as the binaries and packages needed to support the traffic simulator used in modeling the dynamics of traffic networks: SUMO.

If you have not done so already, download the Flow github repository.

```
git clone https://github.com/flow-project/flow.git
cd flow
```

We begin by creating a conda environment and installing Flow and its dependencies within the environment. This can be done by running the below script. Be sure to run the below commands from `/path/to/flow`.

```
# create a conda environment
conda env create -f environment.yml
source activate flow
# install flow within the environment
python setup.py develop
```

Next, we install the necessary SUMO binaries and python tools. The below commands walk you through installing and building SUMO locally. Note that if this does not work, you are recommended to point an issue on the flow-dev

message board or refer to SUMO's documentation regarding installing their software. In additional, if you are interested in trying a simpler and faster set of experimental setup instructions involving installing pre-compiled binaries, we refer you to section 1.c.

We begin by downloading SUMO's github directory:

```
cd ~
git clone https://github.com/eclipse/sumo.git
cd sumo
git checkout 1d4338ab80
make -f Makefile.cvs
```

If you have OSX, run the following commands. If you don't have brew you can find installation instructions at <https://docs.brew.sh/Installation>

```
brew update
brew install Caskroom/cask/xquartz
brew install autoconf
brew install automake
brew install pkg-config
brew install libtool
brew install gdal
brew install proj
brew install xerces-c
brew install fox
export CPPFLAGS=-I/opt/X11/include
export LDFLAGS=-L/opt/X11/lib
./configure CXX=clang++ CXXFLAGS="-stdlib=libc++ -std=gnu++11" --with-xerces=/usr/
↪local --with-proj-gdal=/usr/local
make -j$nproc
echo 'export SUMO_HOME="$HOME/sumo"' >> ~/.bash_profile
echo 'export PATH="$HOME/sumo/bin:$PATH"' >> ~/.bash_profile
echo 'export PYTHONPATH="$HOME/sumo/tools:$PYTHONPATH"' >> ~/.bash_profile
source ~/.bash_profile
```

If you have Ubuntu 14.04+, run the following command

```
./configure
make -j$nproc
echo 'export SUMO_HOME="$HOME/sumo"' >> ~/.bashrc
echo 'export PATH="$HOME/sumo/bin:$PATH"' >> ~/.bashrc
echo 'export PYTHONPATH="$HOME/sumo/tools:$PYTHONPATH"' >> ~/.bashrc
source ~/.bashrc
```

Finally, test your SUMO install and version by running the following commands

```
which sumo
sumo --version
sumo-gui
```

If you are a mac user and the above command gives you the error `FXApp:openDisplay`: unable to open display :0.0 make sure to open the application XQuartz.

2.2 b. Testing the Installation

Once the above modules have been successfully installed, we can test the installation by running a few examples. Before trying to run any examples, be sure to enter your conda environment by typing:

```
source activate flow
```

Let's see some traffic action:

```
python examples/sumo/sugiyama.py
```

Running the following should result in the loading of the SUMO GUI. Click the run button and you should see unstable traffic form after a few seconds, a la (Sugiyama et al, 2008). This means that you have Flow properly configured with SUMO and Flow!

2.3 c. Testing your installation (Ray RLlib)

See getting started with RLlib for sample commands.

To run any of the RL examples, make sure to run

```
source activate flow
```

In order to test run an Flow experiment in RLlib, try the following command:

```
python examples/rllib/stabilizing_the_ring.py
```

If it does not fail, this means that you have Flow properly configured with RLlib.

To visualize the training progress:

```
tensorboard --logdir=~/ray_results
```

If tensorboard is not installed, you can install with pip:

```
pip install tensorboard
```

For information on how to deploy a cluster, refer to the Ray instructions. The basic workflow is running the following locally, ssh-ing into the host machine, and starting jobs from there.

```
ray create_or_update scripts/ray_autoscale.yaml
ray teardown scripts/ray_autoscale.yaml
```

2.4 d. Testing your installation (rllab-multiagent)

To run any of the RL examples, make sure to run

```
source activate flow
```

In order to test run an Flow experiment in rllab-multiagent, try the following command:

```
python examples/rllab/stabilizing_the_ring.py
```

If it does not fail, this means that you have Flow properly configured with rllab-multiagent.

2.5 e. Easy Install SUMO (optional)

In this section, we present and faster and simpler method of installing the necessary SUMO binaries and python tools. These setup instructions are still experimental, so any and all feedback is greatly appreciated!

In order to install everything you will need from SUMO, run one of the below scripts from the Flow main directory. Choose the script that matches the operating system you are running.

For Ubuntu 14.04:

```
scripts/setup_sumo_ubuntu1404.sh
```

For Ubuntu 16.04:

```
scripts/setup_sumo_ubuntu1604.sh
```

For Ubuntu 18.04:

```
scripts/setup_sumo_ubuntu1804.sh
```

For Mac:

```
scripts/setup_sumo_osx.sh
```

If you are using an unsupported operating system (e.g. Arch Linux), or the binaries provided by the above scripts are no compatible with your machine, you will have to personally build the SUMO binary files. For more, please see section 1.a or refer to SUMO's documentation.

2.6 f. Rllab-multiagent (optional)

Flow has been tested on a variety of RL libraries, the installation of which is optional but may be of use when trying to execute some of the examples files located in Flow. rllab-multiagent is one of these such libraries. In order to install the *rllab-multiagent* library, follow the below instructions

```
cd ~
git clone https://github.com/cathywu/rllab-multiagent.git
cd rllab-multiagent
python setup.py develop
```

For linux run

```
echo 'export PYTHONPATH="$HOME/rllab-multiagent:$PYTHONPATH"' >> ~/.bashrc
source ~/.bashrc
```

For mac run

```
echo 'export PYTHONPATH="$HOME/rllab-multiagent:$PYTHONPATH"' >> ~/.bash_profile
source ~/.bash_profile
```

2.7 g. Ray/RLlib (optional)

RLlib is another RL library that has been extensively tested on the Flow repository. First visit <https://github.com/eugenevinitsky/ray/blob/master/doc/source/installation.rst> and install the required packages. Do NOT *pip install ray*.

The installation process for this library is as follows:

```
cd ~
git clone https://github.com/eugenevinitsky/ray.git
cd ray/python/
python setup.py develop
```

If missing libraries cause errors, please also install additional required libraries as specified at <http://ray.readthedocs.io/en/latest/installation.html> and then follow the setup instructions.

2.8 h. Custom configuration

You may define user-specific config parameters as follows

```
cp flow/core/config.template.py flow/core/config.py  # Create template for users␣
↪using pycharm
```

CHAPTER 3

Remote installation using docker

3.1 a. Installation

Installation of a remote desktop and docker to get access to flow quickly

First install docker: https://www.docker.com/

In terminal

```
1° docker pull lucasfischerberkeley/flowdesktop
2° docker run -d -p 5901:5901 -p 6901:6901 -p 8888:8888 lucasfischerberkeley/
↪flowdesktop
```

Go into your browser (Firefox, Chrome, Safari)

```
1° Go to http://localhost:6901/?password=vncpassword
2° Go to Applications and open Terminal Emulator
3° For SUMO: Write python flow/examples/sumo/sugiyama.py and run it
4° For rllib : Write python flow/examples/rllib/stabilizing_the_ring.py and run it
5° For rllab : source activate flow-rllab and python flow/examples/rllab/figure_eight.
↪py ( first time, run it twice)
```

3.2 b. Notebooks and tutorial

In the docker desktop

```
1° Go into Terminal Emulator
2° Run jupyter notebook --NotebookApp.token=admin --ip 0.0.0.0 --allow-root
```

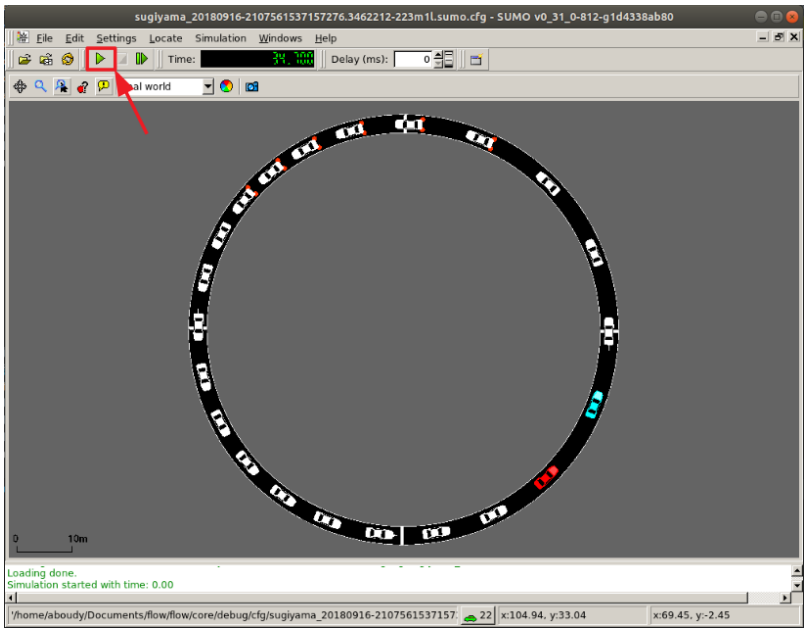
Go into your browser (Firefox, Chrome, Safari)

```
1° go to localhost:8888/tree
2° the password is 'admin' and you can run all your notebooks and tutorials
```

CHAPTER 4

Visualization

Flow supports visualization of both rllab and RLlib computational experiments. When using one of the below visualizers, a window will appear similar to the one in the figure below. Click on the play button (highlighted in red) and the simulation will begin, with the autonomous vehicles exhibiting the behavior trained by the reinforcement learning algorithm.



4.1 rllab

Call the rllab visualizer with

```
python ./visualizer_rllab.py /result_dir/itr_XXX.pkl
```

The rllab visualizer also takes some inputs:

- `--num_rollouts`
- `--plotname`
- `--use_sumogui`
- `--run_long`
- `--emission_to_csv`

The `params.pkl` file can be used as well.

4.2 RLlib

Call the RLlib visualizer with

```
python ./visualizer_rllib.py /ray_results/result_dir 1
# OR
python ./visualizer_rllib.py /ray_results/result_dir 1 --run PPO
# OR
python ./visualizer_rllib.py /ray_results/result_dir 1 --run PPO \
    --module cooperative_merge --flowenv TwoLoopsMergePOEnv \
    --exp_tag cooperative_merge_example
```

The first command-line argument corresponds to the directory containing experiment results (usually within RLlib's `ray_results`). The second is the checkpoint number, corresponding to the iteration number you wish to visualize. The `--run` input is optional; the default algorithm used is PPO. If the experiment module, Flow environment name, and experiment tag have not been stored automatically (see section below), then those parameters can be passed in using the flags `--module`, `--flowenv`, and `--exp_tag`.

4.2.1 Parameter storage

RLlib doesn't automatically store all parameters needed for restoring the state of a Flow experiment upon visualization. As such, Flow experiments in RLlib include code to store relevant parameters. Include the following code snippet in RLlib experiments you will need to visualize

```
# Logging out flow_params to ray's experiment result folder
json_out_file = alg.logdir + '/flow_params.json'
with open(json_out_file, 'w') as outfile:
    json.dump(flow_params, outfile, cls=NameEncoder, sort_keys=True, indent=4)
```

These lines should be placed after initialization of the PPOAgent RL algorithm as it relies on `alg.logdir`. Store parameters before training, though, so partially-trained experiments can be visualized.

Another thing to keep in mind is that Flow parameters in RLlib experiments should be defined **outside** of the `make_create_env` function. This allows that environment creator function to use other experiment parameters later, upon visualization.

CHAPTER 5

Code Documentation

5.1 flow package

5.1.1 Subpackages

flow.benchmarks package

Submodules

flow.benchmarks.bottleneck0 module

flow.benchmarks.bottleneck1 module

flow.benchmarks.bottleneck2 module

flow.benchmarks.figureeight0 module

flow.benchmarks.figureeight1 module

flow.benchmarks.figureeight2 module

flow.benchmarks.grid0 module

flow.benchmarks.grid1 module

flow.benchmarks.merge0 module

flow.benchmarks.merge1 module

flow.benchmarks.merge2 module

Module contents

flow.controllers package

CHAPTER 6

Indices and tables

- genindex
- modindex
- search