Practical Tutorial on Using Reinforcement Learning Algorithms for Continuous Control

Reinforcement Learning Summer School 2017

Peter Henderson Riashat Islam

McGill

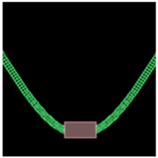
3rd July 2017

Deep Reinforcement Learning

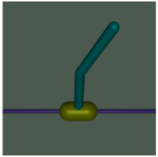
Classical Control Tasks

Cart Pole Balancing
+
Inverted Pendulum

Acrobot



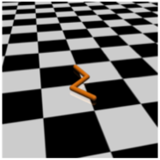
Mountain Car

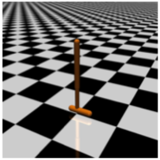


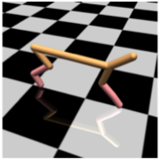
Double Inverted Pendulum

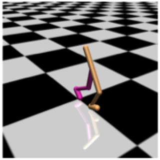
Deep Reinforcement Learning

Locomotion Tasks


Swimmer


Hopper


Half Cheetah


Walker


Ant


Simplified Humanoid


Full Humanoid

Outline

1. Introduction to rllab toolkit - motivation and design
2. Setting up openai/rllab and MuJoCo simulator
3. Walk through example of policy gradient algorithms
    - Deep Deterministic Policy Gradient (DDPG)
    - Trust Region Policy Optimization (TRPO)
4. Live demo and exercises
5. Building your own environment

All the material is available online : `https://github.com/Breakend/RLSSContinuousControlTutorial`

OpenAI RLLAB

Continuous Control Tasks

- High dimensional continuous action space
- Open source implementations of policy gradient algorithms
- Batch gradient-based algorithms
  - REINFORCE [Williams, 1992]
  - TRPO [Schulman et al.2015]
- Online algorithms
  - DDPG [Lillicrap et al. 2015]

...and many others

Available Modules

- Policy Networks
  - Categorical MLP
  - Deterministic MLP
  - Gaussian MLP
- Q Networks
  - Continuous MLP Q Function
- Optimizers
  - First Order
  - Conjugate Gradient
  - Hessian Free
  - LBFGS
- Exploration Strategies
  - Gaussian Exploration
  - OU Strategy

Setting up rllab and MuJoCo Simulator

http://www.mujoco.org/
https://conda.io/miniconda.html
https://github.com/openai/rllab

Secure https://www.roboti.us/license.html

MuJoCo *advanced physics simulation*

Home Gallery Benchmark Download License Changelist Contact Documentation Forum

MuJoCo 1.50 was released on April 23, 2017. Student licenses are now free.

MuJoCo Pro Trial License: 30 days

We invite you to register for a free trial of MuJoCo Pro. Trials are limited to one per user per year. After registration you will receive an email with your activation key and license text. The activati
Download the 'getid' executable corresponding to your platform (using the links below) and run it to obtain your Computer id.

Full name		
Email address		
Computer id		Win32 Win64 Linux OSX
Acceptance	I agree to the terms and conditions of the Trial License.	

Submit

Within the 30-day trial period you may request up to 3 activation keys for different computers or operating systems. All keys will expire on the same date. If you accidentally submit the sam activation key without counting it towards the limit.

MuJoCo Pro Personal License: 1 year

There are three types of personal license: Student, Non-commercial and Commercial. Compared to institutional licenses, the personal license agreements are simpler (see links below) and do representative, however the activation keys are hardware-locked. The licensed user may use the software on up to 3 computers for Non-commercial and Commercial licenses, and one computer f

The annual license fees and use restrictions are summarized below. For customers in countries whose GDP(PPP)-per-capita is less than 50% of the USA, we offer a 50% discount.

Personal Student	Free	Available to full-time students for use in personal projects and class work only. The software may not be used as part of employment, or in projects receivin Requires school email address.
Personal Non-commercial	$500	Available to anyone for use in personal projects. Also available to students, faculty and staff at academic institutions for use in research and education.
Personal Commercial	$2,000	Available to anyone for use in any project.

Secure https://www.roboti.us/index.html

MuJoCo *advanced physics simulation*

Home Gallery Benchmark Download License Changelist Contact Documentation Forum

MuJoCo 1.50 was released on April 23, 2017. Student licenses are now free.

MuJoCo Pro

MuJoCo Pro is a dynamic library with C/C++ API. It is intended for researchers and developers with computational background. It includes an XML parser, model compiler, simulator, and

Commercial license. Compatible with 32-bit and 64-bit Windows, 64-bit Linux and OSX.

MuJoCo Pro requires an **activation key** which is provided to licensed users. See License page.

Download			
	mjpro150 win32	mjpro140 win32	mjpro131 win32
	mjpro150 win64	mjpro140 win64	mjpro131 win64
	mjpro150 linux	mjpro140 linux	mjpro131 linux
	mjpro150 osx	mjpro140 osx	mjpro131 osx

MuJoCo HAPTIX

MuJoCo HAPTIX is an end-user product with full-featured GUI. It has a socket-based API exposing a subset of the functions and data structures available in the Pro library. MuJo simulator, or as a simulator customized for the needs of the DARPA Hand Proprioception & Touch Interfaces (HAPTIX) program. To achieve the latter goal, it integrates real-time motion of a simulated prosthetic hand as well as track the user's head and implement a stereoscopic virtual environment.

Free license. Compatible with 64-bit Windows only.

Download mjhaptix150 mjhaptix140 mjhaptix131

MuJoCo VR

MuJoCo VR is a standalone simulator allowing the user to interact with MuJoCo models in VR. Only the HTC Vive is currently supported, but support for the Oculus Rift will soon b simulator is available as a code sample in MuJoCo Pro, and can be compiled for Windows, Linux and OSX. The precompiled executable available here is Windows only.

Free license. Compatible with 64-bit Windows only.

Download mjvr140 beta1

Secure https://conda.io/miniconda.html

Conda

Miniconda

	Windows	Mac OS X
Python 3.6	64-bit (exe installer) 32-bit (exe installer)	64-bit (bash installer)
Python 2.7	64-bit (exe installer) 32-bit (exe installer)	64-bit (bash installer)

We also offer Miniconda with Python 3.6 for Power8 and Miniconda with Python 2.7 for Power8.

See the Quick install page for installation instructions.

MD5 sums for the downloads can be found here.

See the change log for conda for a list of changes.

These Miniconda installers contain the conda package manager and Python. Once Miniconda is installed, you can use the conda command to install any other packages and create environm

```
$ conda install numpy
...
$ conda create -n py3k anaconda python=3
...
```

There are two variants of the installer: Miniconda is Python 2 based and Miniconda3 is Python 3 based. Note that the choice of which Miniconda is installed only affects the root environment you install, you can still install both Python 2.x and Python 3.x environments.

The other difference is that the Python 3 version of Miniconda will default to Python 3 when creating new environments and building packages. So for instance, the behavior of

```
$ conda create -n myenv python
```

will be to install Python 2.7 with the Python 2 Miniconda and to install Python 3.6 with the Python 3 Miniconda. You can override the default by explicitly setting `python=2` or `python=3`. It als when using `conda build`.

Installation
Preparation
Express Install
Manual Install
Running Experiments
Integrating with OpenAI Gym
Implementing New Environments
Implementing New Algorithms (Basic)
Implementing New Algorithms (Advanced)
Running jobs on EC2

Installation

Preparation

You need to edit your `PYTHONPATH` to include the rllab directory:

```
export PYTHONPATH=path_to_rllab:$PYTHONPATH
```

Express Install

The fastest way to set up dependencies for rllab is via running the setup script.

- On Linux, run the following:

```
./scripts/setup_linux.sh
```

- On Mac OS X, run the following:

```
./scripts/setup_osx.sh
```

The script sets up a conda environment, which is similar to `virtualenv`. To start using it, run the following:

```
source activate rllab3
```

Optionally, if you would like to run experiments that depends on the Mujoco environment, you can set it up by running the following command:

```
./scripts/setup_mujoco.sh
```

and follow the instructions. You need to have the zip file for Mujoco v1.31 and the license file ready.

Walk Through Example

Trust Region Policy Optimization (TRPO)

Running TRPO Algorithm

```python
from rllab.envs.gym_env import GymEnv
#using environments directly from OpenAI Gym

env = TfEnv(normalize(gymenv))
policy = GaussianMLPPolicy()
baseline = LinearFeatureBaseline(env_spec=env.spec)

#TRPO class
algo = TRPO(env=env,
            policy=policy,
            baseline=baseline,
            optimizer = ConjugateGradientOptimizer())

#train networks using trpo algorithm
run_experiment_lite(algo.train())
```

TRPO - Collect Samples

```python
class BatchSampler(BaseSampler):

def obtain_samples(self, itr):
    cur_params = self.algo.policy.get_param_values()
    paths = parallel_sampler.sample_paths(
        policy_params=cur_params,
        max_samples=self.algo.batch_size,
        max_path_length=self.algo.max_path_length,
        scope=self.algo.scope)
```

TRPO - Optimize Policy

```
def train(self):
    for itr in range(current_itr, n_itr):
        paths = sampler.obtain_samples(itr)
        samples_data = sampler.process_samples(itr, paths)
        optimize_policy(itr, samples_data)
```

```
def optimize_policy(itr, samples_data):
    opt = optimizer
    loss_before = opt.loss(all_input_values)
    mean_kl_before = opt.constraint_val(all_input_values)
    mean_kl = opt.constraint_val(all_input_values)
    loss_after = opt.loss(all_input_values)
```

Let's run the code...

Interactive demo!

Walk Through Example

Deep Deterministic Policy Gradient (DDPG)

Running DDPG Algorithm

```
#define policy network
policy = DeterministicMLPPolicy()
#define exploration strategy
es = OUStrategy(env_spec=env.spec)
#define critic network
qf = ContinuousMLPQFunction()
#ddpg module from rllab.algos
algo = DDPG(env=env,policy=policy,es=es,qf=qf)
#train networks
run_experiment_lite(algo.train())
```

```
#run experiment from command line:
python run_ddpg.py Hopper-v1 --num_epochs 10000
```

DDPG Code Snippets - Samples to Replay Buffer

```
#exploration strategy
es = OUStrategy(env_spec=env.spec)
#action from current policy
action = es.get_action(itr, obs, policy)
#take step in environment
next_obs, rwd, terminal = env.step(action)
#add samples to replay buffer
pool.add_sample(obs, action, rwd, terminal)

#fill up replay buffer with off-policy samples
#sample random minibatch
#train the actor and critic networks
if pool.size >= self.min_pool_size:
    # Train policy
    batch = pool.random_batch(self.batch_size)
    do_training(itr, batch)
```

DDPG Code Snippets - Train Actor/Critic Networks

```
def do_training(itr, batch):
    next_actions, _ = target_policy.get_actions(next_obs)
    next_qvals = target_qf.get_qval(next_obs, next_actions)

    ys = rewards + discount * next_qvals
    qf_loss, qval = f_train_qf(ys, obs, actions)
    policy_surr = f_train_policy(obs)
```

Let's run the code...

Interactive demo!

Evaluating Performance

- Plotting results
- Video demonstrations of learned behaviour

Setting Up Parameters and Logging Directory

```python
# Record Gym environment logs and videos (cubic schedule)
gymenv = GymEnv(args.env,
                force_reset=True,
                record_video=True,
                record_log=True)

run_experiment_lite(
    algo.train(),
    # Where to store logs, params, videos, and csv output
    log_dir=args.data_dir,
    # Number of parallel workers for sampling
    n_parallel=1,
    # Only keep the latest parameters
    snapshot_mode="last",
    # random seed
    seed=1,
    # ec2, local, or docker modes
    mode="ec2" if args.use_ec2 else "local",
)
```

Let's run the code...

Let's see how our earlier runs are doing? Do we have some videos?

Building your own environment

You can create custom environments to suit your needs and make more complex tasks!

Modifying a Gym Env

```
class GravityEnv(HopperEnv, utils.EzPickle):
    """
    Allows the gravity to be changed by the
    """
    def __init__(
            self,
            gravity=-9.81,
            *args,
            **kwargs):
        HopperEnv.__init__(self)
        utils.EzPickle.__init__(self)

        self.model.opt.gravity = (mujoco_py.mjtypes.c_double * 3) \
                                    (*[0., 0., gravity])
        self.model._compute_subtree()
        self.model.forward()
```

Registering a new Gym Env

```
arg_dict = dict(id="HopperHalfGravity-v0",
                entry_point="modified_gravity_hopper:GravityEnv"
                max_episode_steps=1000,
                kwargs={"gravity" : -1.0})

gym.envs.register(**arg_dict)
```

Let's run the code...

Interactive demo!

gym-extensions : A Home for Multi-task Learning Envs (pull requests welcome!)

GitHub, Inc. [US] https://github.com/Breakend/gym-extensions

README.md

McGill

gym-extensions

This repo is intended as an extension for OpenAI Gym for auxiliary tasks (multitask learning, transfer learning, inverse reinforcement learning, etc.)

Install

Currently we are working on a pip install process to make this easier, but for now you can simply do

```
git clone https://github.com/Breakend/gym-extensions-multitask.git
export PYTHONPATH="${PYTHONPATH}:./gym-extensions-multitask"
python
>>> import gym_extensions
```

More info

More information will be provided on our doc website: https://breakend.github.io/gym-extensions/

Contributions

For contributing environments please use the general directory structure we have in place and provide **pull requests**. We're still working on making this extension to OpenAI gym the best possible so things may change. Any changes to existing environments should involve an incremental update to the name of the environment (i.e. Hopper-v0 vs. Hopper-v1). If you are not associated with McGill and contribute significantly, please add your association to:

```
docs/index.md
```

Thank You

The only stupid question is the one you were afraid to ask but never did - Rich Sutton

- rllab framework
  `https://github.com/openai/rllab`
- MuJoCo
  `http://www.mujoco.org/`
- Roboschool (open-source alternative to MuJoCo)
  `https://github.com/openai/roboschool`
- Gym
  `https://github.com/openai/gym`
- gym-extensions
  `https://github.com/Breakend/gym-extensions`
- Our Slides and Code
  `https://github.com/Breakend/RLSSContinuousControlTutorial`