RLFlow Documentation

Release 0.1.3

Trevor Barron

Mar 15, 2017

Contents

Contents:

CHAPTER 1

RLFlow

A framework for learning about and experimenting with reinforcement learning algorithms. It is built on top of TensorFlow and interfaces with OpenAI gym (universe should work, too). It aims to be as modular as possible so that new algorithms and ideas can easily be tested. I started it to gain a better understanding of core RL algorithms and maybe it can be useful for others as well.

Features

Algorithms (future algorithms italicized):

- MDP algorithms
  - Value iteration
  - Policy iteration
- Temporal Difference Learning
  - SARSA
  - Deep Q-Learning
  - *Policy gradient Q-learning*
- Gradient algorithms
  - Vanilla policy gradient
  - *Deterministic policy gradient*
  - *Natural policy gradient*
- Gradient-Free algorithms
  - *Cross entropy method*

Function approximators (defined by TFLearn model):

- Linear

- Neural network
- *RBF*

Works with any OpenAI gym environment.

Future Enhancements

- Improved TensorBoard logging
- Improved model snapshotting to include exploration states, memories, etc.
- Any suggestions?

Fixes

- Errors / warnings on TensorFlow session save

License

- Free software: MIT license
- Documentation: https://rlflow.readthedocs.io.

CHAPTER 2

Installation

Stable release

To install RLFlow, run this command in your terminal:

```
$ pip install rlflow
```

This is the preferred method to install RLFlow, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for RLFlow can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/tpbarron/rlflow
```

Or download the tarball:

```
$ curl  -OL https://github.com/tpbarron/rlflow/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Getting Started

To use RLFlow in a project:

```
from __future__ import print_function

import gym
import tensorflow as tf

from rlflow.core import tf_utils
from rlflow.policies.f_approx import Network
from rlflow.algos.grad import PolicyGradient


def build_network(name_scope, env):
    w_init_dense = tf.truncated_normal_initializer() #contrib.layers.xavier_
↪initializer()
    b_init = tf.constant_initializer(value=0.0)

    with tf.variable_scope(name_scope):
        input_tensor = tf.placeholder(tf.float32,
                                      shape=tf_utils.get_input_tensor_shape(env),
                                      name='policy_input_'+name_scope)
        net = tf.contrib.layers.fully_connected(input_tensor,
                                                32, #env.action_space.n, #32,
                                                activation_fn=tf.nn.tanh, #sigmoid,
                                                weights_initializer=w_init_dense,
                                                biases_initializer=b_init,
                                                scope='dense1_'+name_scope)
        net = tf.contrib.layers.fully_connected(net,
                                                env.action_space.n,
                                                weights_initializer=w_init_dense,
                                                biases_initializer=b_init,
                                                scope='dense2_'+name_scope)
        net = tf.contrib.layers.softmax(net)

    return [input_tensor], [net]
```

```python
if __name__ == "__main__":
    # Create the desired environment
    env = gym.make("CartPole-v0")

    # Set up the network we want to use. In this case it is a simple
    # linear model but can be an arbitrary structure, just be sure the
    # inputs and outputs are proper. Here we use softmax outputs since
    # we want to sample from them as probabilities.
    inputs, outputs = build_network("train_policy", env)

    # Create the approximator object. This is just and abstraction of the
    # model structure
    policy = Network(inputs, outputs, scope="train_policy")

    # Now instantiate our algorithm, a basic policy gradient implementation.
    pg = PolicyGradient(env,
                        policy,
                        episode_len=500,
                        discount=0.99,
                        optimizer=tf.train.AdamOptimizer(learning_rate=0.005))

    # Run the algorithm for a desired number of episodes. In this call
    # one can also specify whether to record data to upload to the
    # OpenAI gym evaluation system.
    pg.train(max_episodes=5000,
             save_frequency=10,
             render_train=True)

    # We could restore a previous model if desired
    # pg.restore(ckpt_file="/tmp/rlflow/model.ckpt-###")

    # Now just test what we have learned!
    rewards = pg.test(episodes=10,
                      record_experience=True)
    print ("Average: ", float(sum(rewards)) / len(rewards))
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/tpbarron/rlflow/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

RLFlow could always use more documentation, whether as part of the official RLFlow docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/tpbarron/rlflow/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *rlflow* for local development.

1. Fork the *rlflow* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/rlflow.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv rlflow
   $ cd rlflow/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 rlflow tests
   $ python setup.py test or py.test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7 and 3.5. Check https://travis-ci.org/tpbarron/rlflow/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_rlflow
```

CHAPTER 5

Authors

- Trevor Barron <barron.trevor@gmail.com>

Contributors

None yet. Why not be the first?

CHAPTER 6

History

0.1.4 (????-??-??)

- Either a TFLearn optimizer object or a string key can be passed to an algorithm
- Fix replay memory bug where size would only reach max size - 1
- Improve discount_rewards efficiency with scipy.signal.lfilter
- TODO: improve session management so user doesn't have to create session
- TODO: implement policy duplicate function, where the duplicated ops are in the same graph with a different scope

0.1.2/3 (2016-12-17)

- Improving meta data and fixing __init__ scripts to load subpackages properly

0.1.0 (2016-12-16)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search