An MDP environment for the OpenAI Gym

Andreas Kirsch blackhc@gmail.com

Abstract

The OpenAI Gym provides researchers and enthusiasts with simple to use environments for reinforcement learning. Even the simplest environment have a level of complexity that can obfuscate the inner workings of RL approaches and make debugging difficult. This whitepaper describes a Python framework that makes it very easy to create simple Markov-Decision-Process environments programmatically by specifying state transitions and rewards of deterministic and non-deterministic MDPs in a domain-specific language in Python. It then presents results and visualizations created with this MDP framework.

Introduction

In reinforcement learning[6], agents learn to maximize accumulated rewards from an environment that they can interact with by observing and taking actions. Usually, these environments satisfy a Markov property and are treated as *Markov Decision Processes* (*MDPs*)[4].

The OpenAI Gym[1] is a standardized and open framework that provides many different environments to train agents against through a simple API.

Even the simplest of these environments already has a level of complexity that is interesting for research but can make it hard to track down bugs. However, the gym provides four very simple environments that are useful for testing. The `gym.envs.debugging` package contains a one-round environment with deterministic rewards and one with non-deterministic rewards, and a two-round environment with deterministic rewards and another one with non-deterministic rewards. The author has found these environments very useful for smoke-testing code changes.

This whitepaper introduces a Python framework that makes it very easy to specify simple MDPs like the ones described above in an extensible way. With it, one can validate that agents converge correctly as well as examine other properties.

MDP framework

Specification of MDPs

MDPs are Markov processes that are augmented with a reward function and discount factor. An MDP can be fully specified by a tuple of:

- a finite set of states,
- a finite set of actions,
- a matrix that specifies probabilities of transitions to a new state for a given a state and action,
- a reward function that specifies the reward for a given action taken in a certain state, and
- a discount rate.

The reward function can be either deterministic, or it can be a probability distribution.

Within the framework, MDPs can be specified in Python using a simple *domain-specific language* (*DSL*). For example, the one-round deterministic environment defined in `gym.envs.debugging.one_round_deterministic_reward` could be specified as follows:

```
from blackhc.mdp import dsl

start = dsl.state()
end = dsl.terminal_state()

action_0 = dsl.action()
action_1 = dsl.action()

start & (action_0 | action_1) > end
start & action_1 > dsl.reward(1.)
```

The DSL is based on the following grammar (using EBNF[5]):

```
TRANSITION ::= STATE '&' ACTION '>' OUTCOME
OUTCOME ::= (REWARD | STATE) ['*' WEIGHT]
ALTERNATIVES ::= ALTERNATIVE ('|' ALTERNATIVE)*
```

For a given state and action, outcomes can be specified. Outcomes are state transitions or rewards. If multiple state transitions or rewards are specified for the same state and action, the MDP is non-deterministic and the state transition (or reward) are determined using a categorical distribution. By default, each outcome is weighted uniformly, except if specified otherwise by either having duplicate transitions or by using an explicit weight factor.

For example, to specify that a state receives a reward of +1 or -1 with equal probability and does not change states with probability 3/4 and only transitions

to the next state with probability 1/4, we could write:

```
state & action > dsl.reward(-1.) | dsl.reward(1.)
state & action > state * 3 | next_state
```

Alternatives are distributive with respect to both conjunctions (`&`) and outcome mappings (`>`), so:

```
(a | b) & (c | d) > (e | f) ===
(a & c > e) | (a & c > f) | (a & d > e) |
(a & d > f) | (b & c > e) | ...
```

Alternatives can consist of states, actions, outcomes, conjunctions or partial transitions. For example, the following are valid alternatives:

```
stateA & actionA | stateB & actionB
(actionA > stateC) | (actionB > stateD)
```

As the DSL is implemented within Python, operator overloading is used to implement the semantics. Operator precedence is favorable as `*` has higher precedence than `&`, which has higher precedence than `|`, which has higher precedence than `>`. This allows for a natural formulation of transitions.

Conventional API

The framework also supports specifying an MDP using a conventional API as DSLs are not always preferred.

```
from blackhc import mdp

spec = mdp.MDPSpec()
start = spec.state('start')
end = spec.state('end', terminal_state=True)
action_0 = spec.action()
action_1 = spec.action()

spec.transition(start, action_0, mdp.NextState(end))
spec.transition(start, action_1, mdp.NextState(end))
spec.transition(start, action_1, mdp.Reward(1))
```

Visualization

To make debugging easier, MDPs can be converted to `networkx` graphs[3] and rendered using `pydotplus` and `GraphViz`[2].

```
from blackhc import mdp
from blackhc.mdp import example
```

```
spec = example.ONE_ROUND_DMDP

spec_graph = spec.to_graph()
spec_png = mdp.graph_to_png(spec_graph)

mdp.display_mdp(spec)
```

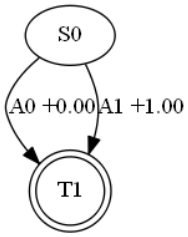

Figure 1: One round deterministic MDP

Optimal values

The framework also contains a small module that can compute the optimal value functions using linear programming.

```
from blackhc.mdp import lp
from blackhc.mdp import example

solver = lp.LinearProgramming(example.ONE_ROUND_DMDP)
print(solver.compute_q_table())
print(solver.compute_v_vector())
```

Gym environment

An environment that is compatible with the OpenAI Gym can be created easily by using the `to_env()` method. It supports rendering into Jupyter notebooks, as RGB array for storing videos, and as png byte data.

```
from blackhc import mdp
from blackhc.mdp import example

env = example.MULTI_ROUND_NMDP.to_env()

env.reset()
env.render()

is_done = False
```

```
while not is_done:
    state, reward, is_done, _ = env.step(env.action_space.sample())
    env.render()
```

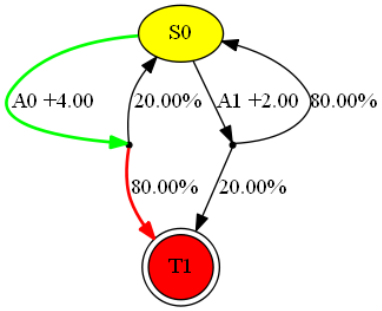

Figure 2: env.render() of `example.MULTI_ROUND_NMDP`

Examples

The `blackhc.mdp.example` package provides 5 MDPs. Four of them match the ones in `gym.envs.debugging`, and the fifth one is depicted in figure 2.

Contribution

A framework to specify MDPs using a domain-specific language in Python was presented. The MDPs can be visualized in Jupyter notebooks and are compatible with the OpenAI Gym.

References

[1] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W. 2016. OpenAI gym. *CoRR*. abs/1606.01540, (2016).

[2] Gansner, E.R. and North, S.C. 2000. An open graph visualization system and its applications to software engineering. *SOFTWARE - PRACTICE AND EXPERIENCE*. 30, 11 (2000), 1203–1233.

[3] Hagberg, A.A., Schult, D.A. and Swart, P.J. 2008. Exploring network structure, dynamics, and function using NetworkX. *Proceedings of the 7th python*

in science conference (sciPy2008) (Pasadena, CA USA, Aug. 2008), 11–15.

[4] Puterman, M.L. 2005. *Markov decision processes : discrete stochastic dynamic programming*. Wiley-Interscience.

[5] Scowen, R.S. 1998. *Extended bNF-a generic base standard*. Technical report, ISO/IEC 14977. http://www.cl.cam.ac.uk/mgk25/iso-14977.pdf.

[6] Sutton, R.S. and Barto, A.G. 1998. *Reinforcement learning : an introduction*. MIT Press.