Prof. Joschka Bödecker, Gabriel Kalweit

Reinforcement Learning
Exercise 2

This week, we provide code snippets that are to be filled by you. Please follow the coding instructions in each task. You will also find tests you can check against.

0 Lecture

Watch *Lecture 03: Model-free Prediction*[1] before the upcoming session on Friday, November 9.

1 Dynamic Programming

The tests for the following tasks are based on the Gridworld environment from Sutton's Reinforcement Learning book chapter 4[2]. The agent moves on an $m \times n$ grid and the goal is to reach one of the terminal states at the top left or the bottom right corner. A visualization can be seen in Figure 1.

$$\begin{bmatrix} T & \cdot & \cdot & \cdot \\ \cdot & A & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & T \end{bmatrix}$$

Figure 1: An example of a 4×4 grid. Terminal states T and agent A.

The agent can go *up*, *down*, *left* and *right*. Actions leading off the edge do not change the state. The agent receives a reward of -1 in each step until it reaches a terminal state. An implementation of this environment is given in `gridworld.py`.

You find the tests in `exercise-02_test.py`. Run them by

```
python exercise-02_test.py -v
```

or by

```
python -m unittest exercise-02_test.py -v.
```

[1] https://ilias.uni-freiburg.de/goto.php?target=xvid_1121347&client_id=unifreiburg
[2] http://incompleteideas.net/book/bookdraft2018mar21.pdf#page=96

1.1 Policy Iteration

(a) Implement the Policy Evaluation function,

`policy_eval(policy, env, discount_factor=1.0, theta=0.00001)`,

in `policy_iteration.py`, where

- `policy` is a $[S, A]$ ($\#S$ states and $\#A$ actions) shaped matrix representing the policy,
- `env` is a discrete OpenAI environment and `env.P[s][a]` is a transition tuple (transition probability, next_state, reward, done) for state s and action a, and
- `theta` is the stopping threshold. We stop the evaluation once our value-function change (difference between two iterations) is less than `theta` for all states.

It returns a vector of length S representing the value-function.

(b) Implement the Policy Improvement function,

`policy_improvement(env, policy_eval_fn=policy_eval, discount_factor=1.0)`,

in `policy_iteration.py`. It returns a tuple `(policy, V)` where `policy` is the optimal policy – a matrix of shape $[S, A]$ where each state s contains a valid probability distribution over actions – and V is the value-function for the optimal policy.

1.2 Value Iteration

(a) Implement the Value Iteration function,

`value_iteration(env, theta=0.0001, discount_factor=1.0)`,

in `value_iteration.py`. It again returns a tuple `(policy, V)` of the optimal policy and the optimal value-function.

(b) What are similarities and differences between Value Iteration and Policy Iteration? Compare the two methods.

2 Experiences

Make a post in thread *Week 02: Planning by Dynamic Programming* in the forum[3], where you provide a brief summary of your experience with this exercise, the corresponding lecture and the last meeting.

[3] `https://ilias.uni-freiburg.de/goto.php?target=frm_1121060&client_id=unifreiburg`