Exercise 2

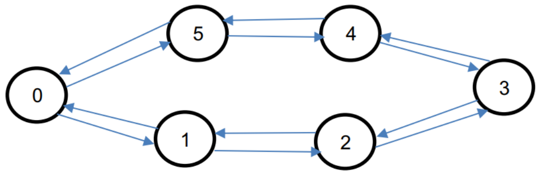
Reinforcement Learning
Lecturer: Alon Peled-Cohen
TA: Yarin Udi

June, 11, 2024

Theory

Question 1

An MDP is represented by a bidirectional cycle of size $2k$, where each node $\in \{0, ..., 2k-1\}$ is a state. The MDP starts at state k. There are two available actions: CW (clockwise) and CCW (counter-clockwise), both are deterministic (CW turns state i into state $i + 1 mod 2k$ and CCW turns every state i to $i - 1 mod 2k$). The reward from each action state (s, a) is 0 for all states but state 0 that receives a (running) reward of 1 (for CW and CCW). The return is discounted with parameter $\gamma < 1$.



1. Write a formal MDP for this problem (i.e., S, A, P, R, s_0).
2. What is the optimal policy π^*?
3. After one iteration of the Value Iteration algorithm (where all states start with a value of 0), which states change their value after one iteration? what are the new values?
4. Which states change their value after two iterations? what are the new values?
5. For $k = 2$ (4 states), calculate $V^*(s)$ for every $s \in S$ (as a function of γ).

Question 2

Consider the following digit game. We have $N+1$ slots, marked by 0 to N. At each round (of $N+1$ rounds) a random digit is chosen (i.e. we chose a random number from $\{0, 1, ..., 9\}$). We can put the digit in one of the unoccupied slots, and that slot becomes occupied. Our aim is to maximize the expected value of the sequence of digits, when viewed as a number (i.e. if we have d_i in slot i then the final value is $\sum_{i=0}^{N} d_i 10^i$).

Example: Suppose $N = 2$. At the beginning, we have three empty slots. Let us denote an empty slot by $\times$, then we have $\times\ \times\ \times$. Suppose the first digit is 5, and we decide to put it in the second slot, then we have $\times 5 \times$. Suppose the second digit is 4 and we decide to put it in the first slot, we have $\times 54$. Finally, the last digit is 4 again, and our final number is 454.

1. Build an MDP for this problem (Hint: you may incorporate the random digit in the state).
2. Show that the optimal policy depends only on the number of empty slots.
3. Compute the optimal policy for $N = 2$ (i.e the number has three digits).

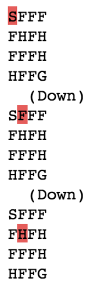
Programming

Intro

The following questions will use the Frozen Lake environment, a simple grid-world MDP that is taken from gym and slightly modified for this assignment. In this MDP, the agent must navigate from the start state to the goal state on a 4x4 grid, with stochastic transitions. You can read more about the environment here: FrozenLake-v1.

You are provided with template Python scripts in which you should implement the algorithms: value_iter.py and policy_iter.py.

An example episode is given below, the agent falls into a hole after two timesteps. Note the stochasticity–on the first step, the DOWN action is selected, but the agent moves to the right.



```
SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
  (Down)
SFFF
FHFH
FFFH
HFFG
```

Question 1: Value Iteration

In this problem, you will implement value iteration, which has the following pseudocode:

```
Value Iteration
  Initialize $V^{(0)}(s) = 0$, for all $s$
  for $i = 0, 1, 2, \ldots$ do
      for for all $s \in S$ do
          $V^{(i+1)}(s) = \max_a \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V^{(i)}(s')]$
```

We additionally define the sequence of greedy policies $\pi^{(0)}, \pi^{(1)}, ...\pi^{(n-1)}$ where

$$\pi^{(i)} = argmax_a \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V^{(i)}(s')] \qquad (1)$$

Your code will return two lists: $[V^{(0)}, V^{(1)}, ..., V^{(n)}]$ and $[\pi^{(0)}, \pi^{(1)}, ..., \pi^{(n-1)}]$.

Note 1: Choose the lower-index action to break ties in $argmax_a$. This will only affect the "# chg actions" printout below – it won't affect the values computed.

Note 2: Make a copy of your value function each iteration and use that copy for the update – don't update your value function in place. Updating in-place is also a valid algorithm, sometimes called Gauss-Seidel value iteration or asynchronous value iteration, but it will cause you to get different results than our reference solution.

1. Implement the relevant code in the attached value_iter.py code. The relevant locations are marked with REPLACE THIS LINE WITH YOUR CODE comment.

2. Submit: (i) plot of the optimal policy at each iteration and (ii) plot of each state value as a function of iterations (different curve for each value, same figure). The script already contains the code for (i).

Submit your modified code, iteration table outputted by the code and plots.

Question 2: Policy Iteration

The next task is to implement exact policy iteration (PI), which has the following pseudocode:

```
Initialize $\pi_0$
for $i = 0, 1, 2, \ldots$ do
    Compute the state-value function $V^{\pi_n}$
    Using $V^{\pi_n}$, compute the state-action-value function $Q^{\pi_n}$
    Compute new policy $\pi_{n+1}(s) = \text{argmax}_a Q^{\pi_n}(s, a)$
```

You will implement the first and second steps of the loop using the attached policy_iter.py. Your code will return two lists: $[V^{(0)}, V^{(1)}, ..., V^{(n)}]$ and $[\pi^{(0)}, \pi^{(1)}, ..., \pi^{(n-1)}]$.

1. Implement the function called compute_vpi that computes the state-value function V^π for an arbitrary policy π . Recall that V^π satisfies the following linear equation:

$$V^\pi(s) = \sum_{s'} P(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')] \tag{2}$$

   You can solve a linear system in your code. (Find an exact solution, e.g., with np.linalg.solve). As before, look for the REPLACE THIS LINE WITH YOUR CODE comment.

2. Implement the function called compute_qpi computes the state-action value function Q^π, defined as follows:

$$Q^\pi(s, a) = \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V^\pi(s')] \tag{3}$$

   As before, look for the REPLACE THIS LINE WITH YOUR CODE comment.

3. Using the functions you implemented in parts (1) and (2) implement the function called policy_iteration which implements policy iteration. This time, look for the YOUR CODE HERE comment.

Submit your modified code, iteration table outputted by the code and value plot.