Machine Learning 2019/2020: Assignment 4 - Reinforcement Learning

Deadline: Friday 6th of December 2019 9pm

First name: Giorgia
Last name: Adorni

About this assignment

In this assignment you will further deepen your understanding of Reinforcement Learning (RL).

Submission instructions

Please write your answers, equations, and code directly in this python notebook and print the final result to pdf (File > Print). Make sure that code has appropriate line breaks such that all code is visible in the final pdf. Also select A3 for the PDF size to prevent content from being clipped.

The final pdf must be named name.lastname.pdf and uploaded to the iCorsi website before the deadline expires. Late submissions will result in 0 points.

Also share this notebook (top right corner 'Share') with teaching.idsia@gmail.com during submission.

Keep your answers brief and respect the sentence limits in each question (answers exceeding the limit are not taken into account).

Learn more about python notebooks and formatting here: https://colab.research.google.com/notebooks/welcome.ipynb

How to get help

We encourage you to use the tutorials to ask questions or to discuss exercises with other students. However, do not look at any report written by others or share your report with others. Violation of that rule will result in 0 points for all students involved. For further questions you can send an email to louis@idsia.ch

1 Basic probability (6p)

Suppose that a migrating lizard that rests in Ticino can be in four different states: Eating (E), Sleeping (S), Fighting (F) and Mating (M), for example protecting its territory against other lizards. Each lizard spends 30% of its time sleeping, 40% eating, 20% fighting and the remaining time mating. A biologist collects a population of lizards and puts them in a cage to study their behaviors. Suppose the probability for a lizard being caught while eating is 0.1, for a sleeping lizard 0.4, for a fighting lizard 0.8 and for the lizards that are mating 0.2, respectively.

Question 1.1 (3p)

What is the relative frequency (probability) for a lizard being caught in the cage?

ANSWER HERE

$$\begin{aligned} P(\text{caught}) &= P(\text{caught}|\text{E})P(\text{E}) + P(\text{caught}|\text{S})P(\text{S}) + P(\text{caught}|\text{F})P(\text{F}) + P(\text{caught}|\text{M})P(\text{M}) = \\ &= 0.1 \cdot 0.4 + 0.4 \cdot 0.3 + 0.8 \cdot 0.2 + 0.2 \cdot 0.1 = \\ &= 0.04 + 0.12 + 0.16 + 0.02 = \\ &= 0.34 \end{aligned}$$

Question 1.2 (3p)

What is the proportion of lizards that are fighting of those that were caught in the cage?

ANSWER HERE

$$P(\text{F}|\text{caught}) \overset{\text{bayes}}{=} \frac{P(\text{caught}|\text{F})P(\text{F})}{P(\text{caught})} = \frac{0.16}{0.34} = 0.47$$

2 Markov Decision Processes (32p)

Suppose a robot is put in a maze with long corridor. The corridor is 1 kilometer long and 5 meters wide. The available actions to the robot are moving forward for 1 meter, moving backward for 1 meter, turning left for 90 degrees and turning right for 90 degrees. If the robot moves and hits the wall, then it will stay in its position and orientation. The robot's goal is to escape from this maze by reaching the end of the long corridor.
Note: the answers in the following questions should not exceed 5 sentences.

Question 2.1 (4p)

Assume the robot receives a +1 reward signal for each time step taken in the maze and +1000 for reaching the final goal (the end of the long corridor). Then you train the robot for a while, but it seems it still does not perform well at all for navigating to the end of the corridor in the maze. What is happening? Is there something wrong with the reward function?

ANSWER HERE

The reward function is wrong. In fact, the robot is not encouraged to reach the end of the corridor, that is the final goal, since it is rewarded for each time step. In this way, for the robot could be better exploring the maze instad of reaching the final goal.

Question 2.2 (4p)

If there is something wrong with the reward function, how could you fix it? If not, how to resolve the training issues?

ANSWER HERE

As I said in the previous answer, the reward function is wrong. It is possible to correct it in the following way:

- not rewarding the robot for each time step or even penalising the time spent in the maze so as to encourage it to reach the end of the maze
- rewarding the robot only for the final goal

Questions 2.3 (2p)

The discounted return for a non-episodic task is defined as

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

where $\gamma \in [0, 1]$ is the discount factor.

Rewrite the above equation such that G_t is on the left hand side and G_{t+1} is on the right hand side.

ANSWER HERE

Since $G_{t+1} = R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \ldots$, G_t can be rewritten as:

$$G_t = R_{t+1} + \gamma\, G_{t+1}$$

Questions 2.4 (2p)

What is the sufficient condition for this infinite series to be a convergent series?

ANSWER HERE

This infinite series is convergent if has a bounded reward sequence and $\gamma < 1$.

Questions 2.5 (5p)

Suppose this infinite series is a convergent series, and each reward in the series is a constant of +1. We know the series is bounded, what is a simple formula for this bound ? Write it down without using summation.

ANSWER HERE

The given infinite series $\sum_{i=0}^{\infty} \gamma^i R_{t+i+1}$ is a geometric series. Since each reward is a constant, it is possible to write $\sum_{i=0}^{\infty} \gamma^i \cdot 1 = \sum_{i=0}^{\infty} \gamma^i$.

Now, supposed thta the series is convergent, and so that $0 \leq \gamma < 1$, it converges to $\frac{1}{1-\gamma}$.

Questions 2.6 (5p)

Let the task be an episodic setting and the robot is running for $T = 5$ time steps. Suppose $\gamma = 0.3$, and the robot receives rewards along the way $R_1 = -1, R_2 = -0.5, R_3 = 2, R_4 = 1, R_5 = 6$. What are the values for $G_0, G_1, G_2, G_3, G_4, G_5$?

ANSWER HERE

$G_0 = R_1 + \gamma\, G_1 = -1 + 0.3 \cdot 0.352 = -0.8944$

$G_1 = R_2 + \gamma\, G_2 = -0.5 + 0.3 \cdot 2.84 = 0.352$

$G_2 = R_3 + \gamma\, G_3 = 2 + 0.3 \cdot 2.8 = 2.84$

$G_3 = R_4 + \gamma\, G_4 = 1 + 0.3 \cdot 6 = 2.8$

$G_4 = R_5 + \gamma\, G_5 = 6 + 0.3 \cdot 0 = 6$

$G_5 = R_6 + \gamma\, G_6 = 0$

G_5 is 0 since $t > T$.

Questions 2.7 (5p)

Suppose each reward in the series is increased by a constant c, i.e. $R_t \leftarrow R_t + c$. Then how does it change G_t ?

ANSWER HERE

$$G_t = \sum_{i=0}^{\infty} \gamma^i (R_{t+1} + c) = \sum_{i=0}^{\infty} \gamma^i R_{t+1} + \sum_{i=0}^{\infty} \gamma^i c = G_t + \sum_{i=0}^{\infty} \gamma^i c = G_t + (c + \gamma c + \gamma^2 c + \gamma^3 c \ldots).$$

Hence, G_t is increased by $\sum_{i=0}^{\infty} \gamma^i c = \frac{c}{1-\gamma}$.

Questions 2.8 (5p)

Now consider episodic tasks, and similar to Question 2.7, we add a constant c to each reward, how does it change G_t ?

ANSWER HERE

$$G_t = \sum_{i=0}^{T-t-1} \gamma^i (R_{t+1} + c) = \sum_{i=0}^{T-t-1} \gamma^i R_{t+1} + \sum_{i=0}^{T-t-1} \gamma^i c = G_t + \sum_{i=0}^{T-t-1} \gamma^i c = G_t + (c + \gamma c + \gamma^2 c + \gamma^3 c \ldots \gamma^{T-t-1} c).$$

Hence, G_t is increased by $\sum_{i=0}^{T-t-1} \gamma^i c = c \cdot \frac{1-\gamma^{T-t}}{1-\gamma} = \frac{c - c \cdot \gamma^{T-t}}{1-\gamma}$, that is the amount of which G_t is increased.

3 Dynamic Programming (62p)

Questions 3.1 (5p)

Write down the Bellman optimality equation for the state value function without using expectation notation, but using probability distributions instead. Define all variables and probability distributions in bullet points.

ANSWER HERE

$$V^*(s) = \max_{\pi} V_{\pi}(s) = \max_{a} \sum_{s',r} p(s', r|s, a)[r + \gamma V^*(s')] \quad \forall s$$

where:

- $a \in A^+$ is an action
- $s \in S^+$ is a state and $s' \in S^+$ is the following state
- $r \in R^+$ is the reward
- $\pi \in (a|s)$ is the policy
- $p(s', r|s, a)$ is the probability of each possible pair of next state an reward given any state and action
- $s \in S^+$ is a state and $s' \in S^+$ is the following state
- γ is the discount factor
- $V_{\pi}(s)$ is the state-value function for policy π and state s
- $V^*(s')$ is the optimal state value function of the following state

Questions 3.2 (5p)

Write down the Bellman optimality equation for the state-action value function without using expectation notation, but using probability distributions instead. Define all variables and probability distributions in bullet points.

ANSWER HERE

$$Q^*(s,a) = \max_\pi Q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} Q^*(s',a')] \quad \forall s$$

where:

- $a \in A^+$ is an action and $a' \in A^+$ is the following action
- $s \in S^+$ is a state and $s' \in S^+$ is the following state
- $r \in R^+$ is the reward
- $\pi \in (a|s)$ is the policy
- $p(s',r|s,a)$ is the probability of each possible pair of next state an reward given any state and action
- γ is the discount factor
- $Q_\pi(s',a')$ is the action-value function for policy π
- $Q^*(s',a')$ is the optimal action-value function for state s' and action a'

Double-click (or enter) to edit

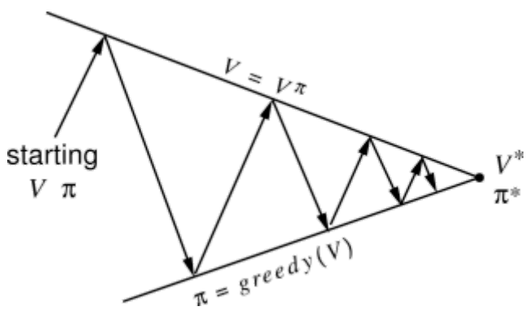
Questions 3.3 (15p)

Consider a 4x4 gridworld depicted in the following table:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

The non-terminal states are $S = \{1, 2, \dots, 14\}$ and the terminal states are $\bar{S} = \{0, 15\}$. There are four available actions for each state, that is $A = \{\text{up}, \text{down}, \text{left}, \text{right}\}$. Assume the state transitions are deterministic and all transitions result in a negative reward −1 (after termination all rewards are zero). If the agent hits the boundary, then its state will remain unchanged, e.g.
$p(s = 8, r = -1|s = 8, a = \text{left}) = 1$. Note: In this exercise, we assume the policy is a deterministic function.

Manually run the policy iteration algorithm (see lecture slide 58) for one iteration. Use the in-place policy iteration algorithm. This means one time policy evaluation with a single pass through the states (16 equations) and one time policy improvement. Assume the initial state value for all 16 cells are 0.0 and the policy initially always outputs the 'left' action. Write down the equations and detailed numerical computations for the updated values of each cell. Use a discount factor $\gamma = 0.5$. Write down the policy after policy improvement.



Read more about this in Sutton & Barto's book http://www.incompleteideas.net/book/ebook/node43.html

---

ANSWER HERE

Policy Evaluation

$$V_\pi(s) = \sum_a \pi(s,a) \sum_{s',r} p(s',r|s,a)[r + \gamma V_\pi(s')]$$

Since the initial policy always outputs the 'left' action, the previous expression is simplified: of the first summation, only a $\pi(s,a)$ term is equal to 1.

$$\begin{aligned}
V_\pi(0) &= 1 \cdot [0 + 0.5 \cdot 0] = 0 && s' = 0 \\
V_\pi(1) &= 1 \cdot [-1 + 0.5 \cdot 0] = -1 && s' = 0 \\
V_\pi(2) &= 1 \cdot [-1 + 0.5 \cdot -1] = -1.5 && s' = 1 \\
V_\pi(3) &= 1 \cdot [-1 + 0.5 \cdot -1.5] = -1.75 && s' = 2 \\
V_\pi(4) &= 1 \cdot [-1 + 0.5 \cdot 0] = -1 && s' = 4 \\
V_\pi(5) &= 1 \cdot [-1 + 0.5 \cdot -1] = -1.5 && s' = 4 \\
V_\pi(6) &= 1 \cdot [-1 + 0.5 \cdot -1.5] = -1.75 && s' = 5 \\
V_\pi(7) &= 1 \cdot [-1 + 0.5 \cdot -1.75] = -1.875 && s' = 6 \\
V_\pi(8) &= 1 \cdot [-1 + 0.5 \cdot 0] = -1 && s' = 8 \\
V_\pi(9) &= 1 \cdot [-1 + 0.5 \cdot -1] = -1.5 && s' = 8 \\
V_\pi(10) &= 1 \cdot [-1 + 0.5 \cdot -1.5] = -1.75 && s' = 9 \\
V_\pi(11) &= 1 \cdot [-1 + 0.5 \cdot -1.75] = -1.875 && s' = 10 \\
V_\pi(12) &= 1 \cdot [-1 + 0.5 \cdot 0] = -1 && s' = 12 \\
V_\pi(13) &= 1 \cdot [-1 + 0.5 \cdot -1] = -1.5 && s' = 12 \\
V_\pi(14) &= 1 \cdot [-1 + 0.5 \cdot -1.5] = -1.75 && s' = 13 \\
V_\pi(15) &= 1 \cdot [0 + 0.5 \cdot 0] = 0 && s' = 15
\end{aligned}$$

Policy Improvement

$$\pi'(s) = \arg\max_a Q_\pi(s,a) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V_\pi(s')]$$

For legibility, in the $\arg\max$ of the following expressions are computed the values with respect to the actions in the following order: up, down, left, right.

$$\begin{aligned}
\pi'(0) &= \arg\max_a \{1(0 + 0.5(V_\pi(0))); \quad 1(0 + 0.5(V_\pi(0))); \quad 1(0 + 0.5(V_\pi(0))); \quad 1(0 + 0.5(V_\pi(0)))\} = \\
&= \arg\max_a \{1(0 + 0.5(0)); \quad 1(0 + 0.5(0)); \quad 1(0 + 0.5(0)); \quad 1(0 + 0.5(V_\pi(0)))\} = \\
&= \arg\max_a \{0; 0; 0; 0\} = \text{ up}
\end{aligned}$$

$$\pi'(1) = \arg\max_a\{1(-1+0.5(V_\pi(1))); \quad 1(-1+0.5(V_\pi(5))); \quad 1(-1+0.5(V_\pi(0))); \quad 1(-1+0.5(V_\pi(2)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1)); \quad 1(-1+0.5(-1.5)); \quad 1(-1+0.5(0)); \quad 1(-1+0.5(-1.5))\} =$$
$$= \arg\max_a\{-1.5; -1.75; -1; -1.75\} = \text{left}$$

$$\pi'(2) = \arg\max_a\{1(-1+0.5(V_\pi(2))); \quad 1(-1+0.5(V_\pi(6))); \quad 1(-1+0.5(V_\pi(1))); \quad 1(-1+0.5(V_\pi(3)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1.5)); \quad 1(-1+0.5(-1.75)); \quad 1(-1+0.5(-1)); \quad 1(-1+0.5(-1.75))\} =$$
$$= \arg\max_a\{-1.75; -1.875; -1.5; -1.875\} = \text{left}$$

$$\pi'(3) = \arg\max_a\{1(-1+0.5(V_\pi(3))); \quad 1(-1+0.5(V_\pi(7))); \quad 1(-1+0.5(V_\pi(2))); \quad 1(-1+0.5(V_\pi(3)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1.75)); \quad 1(-1+0.5(-1.875)); \quad 1(-1+0.5(-1.5)); \quad 1(-1+0.5(-1.75))\} =$$
$$= \arg\max_a\{-1.875; -1.9375; -1.75; -1.875\} = \text{left}$$

$$\pi'(4) = \arg\max_a\{1(-1+0.5(V_\pi(0))); \quad 1(-1+0.5(V_\pi(8))); \quad 1(-1+0.5(V_\pi(4))); \quad 1(-1+0.5(V_\pi(5)))\} =$$
$$= \arg\max_a\{1(-1+0.5(0)); \quad 1(-1+0.5(-1)); \quad 1(-1+0.5(-1)); \quad 1(-1+0.5(-1.5))\} =$$
$$= \arg\max_a\{-1; -1.5; -1.5; -1.75\} = \text{up}$$

$$\pi'(5) = \arg\max_a\{1(-1+0.5(V_\pi(1))); \quad 1(-1+0.5(V_\pi(9))); \quad 1(-1+0.5(V_\pi(4))); \quad 1(-1+0.5(V_\pi(6)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1)); \quad 1(-1+0.5(-1.5)); \quad 1(-1+0.5(-1)); \quad 1(-1+0.5(-1.75))\} =$$
$$= \arg\max_a\{-1.5; -1.875; -1.5; -1.875\} = \text{up}$$

$$\pi'(6) = \arg\max_a\{1(-1+0.5(V_\pi(2))); \quad 1(-1+0.5(V_\pi(10))); \quad 1(-1+0.5(V_\pi(5))); \quad 1(-1+0.5(V_\pi(7)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1.5)); \quad 1(-1+0.5(-1.75)); \quad 1(-1+0.5(-1.5)); \quad 1(-1+0.5(-1.875))\} =$$
$$= \arg\max_a\{-1.75; -1.875; -1.75; -1.9375\} = \text{up}$$

$$\pi'(7) = \arg\max_a\{1(-1+0.5(V_\pi(3))); \quad 1(-1+0.5(V_\pi(11))); \quad 1(-1+0.5(V_\pi(6))); \quad 1(-1+0.5(V_\pi(7)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1.75)); \quad 1(-1+0.5(-1.875)); \quad 1(-1+0.5(-1.75)); \quad 1(-1+0.5(-1.875))\} =$$
$$= \arg\max_a\{-1.875; -1.9375; -1.875; -1.9375\} = \text{up}$$

$$\pi'(8) = \arg\max_a\{1(-1+0.5(V_\pi(4))); \quad 1(-1+0.5(V_\pi(12))); \quad 1(-1+0.5(V_\pi(8))); \quad 1(-1+0.5(V_\pi(9)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1)); \quad 1(-1+0.5(-1)); \quad 1(-1+0.5(-1)); \quad 1(-1+0.5(-1.5))\} =$$
$$= \arg\max_a\{-1.5; -1.875; -1.5; -1.75\} = \text{up}$$

$$\pi'(9) = \arg\max_a\{1(-1+0.5(V_\pi(5))); \quad 1(-1+0.5(V_\pi(13))); \quad 1(-1+0.5(V_\pi(8))); \quad 1(-1+0.5(V_\pi(10)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1.5)); \quad 1(-1+0.5(-1.5)); \quad 1(-1+0.5(-1)); \quad 1(-1+0.5(-1.75))\} =$$
$$= \arg\max_a\{-1.75; -1.75; -1.5; -1.875\} = \text{left}$$

$$\pi'(10) = \arg\max_a\{1(-1+0.5(V_\pi(6))); \quad 1(-1+0.5(V_\pi(14))); \quad 1(-1+0.5(V_\pi(9))); \quad 1(-1+0.5(V_\pi(11)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1.75)); \quad 1(-1+0.5(-1.75)); \quad 1(-1+0.5(-1.5)); \quad 1(-1+0.5(-1.875)\} =$$
$$= \arg\max_a\{-1.875; -1.875; -1.75; -1.9375\} = \text{left}$$

$$\pi'(11) = \arg\max_a\{1(-1+0.5(V_\pi(7))); \quad 1(-1+0.5(V_\pi(15))); \quad 1(-1+0.5(V_\pi(10))); \quad 1(-1+0.5(V_\pi(11)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1.875)); \quad 1(-1+0.5(0)); \quad 1(-1+0.5(-1.75)); \quad 1(-1+0.5(-1.875))\} =$$
$$= \arg\max_a\{-1.9375; -1; -1.875; -1.9375\} = \text{down}$$

$$\pi'(12) = \arg\max_a\{1(-1+0.5(V_\pi(8))); \quad 1(-1+0.5(V_\pi(12))); \quad 1(-1+0.5(V_\pi(12))); \quad 1(-1+0.5(V_\pi(13)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1)); \quad 1(-1+0.5(-1)); \quad 1(-1+0.5(-1)); \quad 1(-1+0.5(-1.5))\} =$$
$$= \arg\max_a\{-1.5; -1.5; -1.5; -1.75\} = \text{up}$$

$$\pi'(13) = \arg\max_a\{1(-1+0.5(V_\pi(9))); \quad 1(-1+0.5(V_\pi(13))); \quad 1(-1+0.5(V_\pi(12))); \quad 1(-1+0.5(V_\pi(14)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1.5); \quad 1(-1+0.5(-1.5)); \quad 1(-1+0.5(-1)); \quad 1(-1+0.5(-1.75)\} =$$
$$= \arg\max_a\{-1.75; -1.75; -1.5; -1.875\} = \text{left}$$

$$\pi'(14) = \arg\max_a\{1(-1+0.5(V_\pi(10))); \quad 1(-1+0.5(V_\pi(14))); \quad 1(-1+0.5(V_\pi(13))); \quad 1(-1+0.5(V_\pi(15)))\} =$$
$$= \arg\max_a\{1(-1+0.5(-1.75)); \quad 1(-1+0.5(-1.75)); \quad 1(-1+0.5(-1.5)); \quad 1(-1+0.5(0))\} =$$
$$= \arg\max_a\{-1.875; -1.875; -1.75; -1\} = \text{right}$$

$$\pi'(15) = \arg\max_a \{1(0 + 0.5(V_\pi(15))); \quad 1(0 + 0.5(V_\pi(15))); \quad 1(0 + 0.5(V_\pi(15))); \quad 1(0 + 0.5(V_\pi(15)))\} =$$
$$= \arg\max_a \{1(0 + 0.5(0)); \quad 1(0 + 0.5(0)); \quad 1(0 + 0.5(0)); \quad 1(0 + 0.5(0))\} =$$
$$= \arg\max_a \{0; 0; 0; 0\} = \text{up}$$

Then,

$$\begin{bmatrix} \uparrow & \leftarrow & \leftarrow & \leftarrow \\ \uparrow & \uparrow & \uparrow & \uparrow \\ \uparrow & \leftarrow & \leftarrow & \downarrow \\ \uparrow & \leftarrow & \rightarrow & \uparrow \end{bmatrix}$$

Questions 3.4 (15p)

Implement the before mentioned environment in the code skeleton below. Come up with your own solution and do not copy the code from a third party source.

Imports

```
import numpy as np
import itertools

np.set_printoptions(precision=3, linewidth=180)
```

Defining the problem

```
class GridWorld:
    UP = 0
    DOWN = 1
    LEFT = 2
    RIGHT = 3

    def __init__(self, side=4):
        self.side = side
        # -------------------------
        # Define integer states, actions, and final states as specified in the problem description

        # TODO insert code here
        self.actions = np.array([self.UP, self.DOWN, self.LEFT, self.RIGHT])
        self.states = np.arange(self.side * self.side)
        self.finals = np.array([0, 15])

        # -------------------------
        self.actions_repr = np.array(['↑', '↓', '←', '→'])

    def reward(self, s, s_next, a):
        # -------------------------
        # Return the reward for the given transition as specified in the problem description

        # TODO insert code here
        reward = -1
        if s in self.finals:
            reward = 0

        return reward

        # -------------------------

    def transition_prob(self, s, s_next, a):
        # -------------------------
        # Return a probability in [0, 1] for the given transition as specified in the problem description

        # TODO insert code here
        prob = 0

        if s in self.finals:
            if s == s_next:
                prob = 1
            else:
                prob = 0
        elif (a == self.UP and (s - self.side == s_next or (s < self.side and s == s_next))) or \
                (a == self.DOWN and (s + self.side == s_next or (s >= self.side * self.side - self.side and s == s_next))) or \
                (a == self.LEFT and ((s % self.side != 0 and s - 1 == s_next) or (s % self.side == 0 and s == s_next))) or \
                (a == self.RIGHT and (((s + 1) % self.side != 0 and s + 1 == s_next) or ((s + 1) % self.side == 0 and s == s_next))):
            prob = 1

        return prob
        # -------------------------

    def print_policy(self, policy):
        P = np.array(policy).reshape(self.side, self.side)
        print(self.actions_repr[P])

    def print_values(self, values):
        V = np.array(values).reshape(self.side, self.side)
        print(V)
```

Questions 3.5 (17p)

Implement policy iteration in the code skeleton below. Come up with your own solution and do not copy the code from a third party source.

Run the code multiple times. Do you always end up with the same policy? Why? (max 4 sentences)

ANSWER HERE

Yes. If the algorithm converges, it converges to the global optimum of the value function, that is unique. The policies are not unique, in fact, in case of a tie, it is possible to obtain the same value function from different policies executing a certain action. In the code implemented below, ties are always solved in the same way: the first action with maximum value is chosen as policy.

Policy iteration

```
def eval_policy(world, policy, values, gamma=0.9, theta=0.01):
    # --------------------------
    # Implement policy evaluation and return the updated value function

    # TODO insert code here
    while True:
        delta = 0.0
        old_values = np.copy(values)

        for s in world.states:
            a = policy[s]
            v = np.zeros_like(world.states, dtype=np.float32)

            for s_next in world.states:
                p = world.transition_prob(s, s_next, a)
                r = world.reward(s, s_next, a)
                v[s_next] = p * (r + gamma * values[s_next])

            values[s] = np.sum(v)
            delta = np.maximum(delta, np.absolute(values[s] - old_values[s]))

        if delta < theta:
            break

    return values
    # --------------------------
```

```
def improve_policy(world, policy, values, gamma=0.9):
    # --------------------------
    # Implement policy improvement and return the updated policy

    # TODO insert code here
    policy_stable = True

    for s in world.states:
        argmax_q = -np.inf
        best_a = None

        for a in world.actions:
            q_actual = 0

            for s_next in world.states:
                p = world.transition_prob(s, s_next, a)
                r = world.reward(s, s_next, a)
                q_actual += p * (r + gamma * values[s_next])

            if q_actual > argmax_q:
                argmax_q = q_actual
                best_a = a

        if policy[s] != best_a:
            policy_stable = False

        policy[s] = best_a

    return policy_stable
    # --------------------------


def policy_iteration(world, gamma=0.9, theta=0.01):
    # Initialize a random policy
    policy = np.array([np.random.choice(world.actions) for s in world.states])
    print('Initial policy')
    world.print_policy(policy)
    # Initialize values to zero
    values = np.zeros_like(world.states, dtype=np.float32)

    # Run policy iteration
    stable = False
    for i in itertools.count():
        print(f'Iteration {i}')
        values = eval_policy(world, policy, values, gamma, theta)
        world.print_values(values)
        stable = improve_policy(world, policy, values, gamma)
        world.print_policy(policy)
        if stable:
            break

    return policy, values


# Evaluate your code, please include the output in your submission
world = GridWorld()
policy, values = policy_iteration(world, gamma=0.5)
```

```
Initial policy
[['←' '↓' '↑' '→']
 ['↑' '←' '↑' '↓']
 ['↑' '↑' '←' '↑']
 ['→' '←' '↓' '↓']]
Iteration 0
[[ 0.    -1.75  -1.992 -1.992]
 [-1.    -1.5   -1.996 -2.   ]
 [-1.5   -1.75  -1.875 -2.   ]
 [-2.    -2.    -1.992  0.   ]]
[['↑' '←' '←' '↑']
 ['↑' '←' '←' '↑']
 ['↑' '↑' '←' '↓']
 ['↑' '↑' '→' '↑']]
Iteration 1
[[ 0.    -1.    -1.5   -1.998]
 [-1.    -1.5   -1.75  -1.999]
 [-1.5   -1.75  -1.875 -1.   ]
 [-1.75  -1.875 -1.     0.   ]]
[['↑' '←' '←' '←']
 ['↑' '↑' '↑' '↓']
 ['↑' '↑' '↓' '↓']
 ['↑' '→' '→' '↑']]
Iteration 2
[[ 0.   -1.   -1.5  -1.75]
 [-1.   -1.5  -1.75 -1.5 ]
 [-1.5  -1.75 -1.5  -1.  ]
 [-1.75 -1.5  -1.    0.  ]]
[['↑' '←' '←' '↓']
 ['↑' '↑' '↑' '↓']
 ['↑' '↑' '↓' '↓']
 ['↑' '→' '→' '↑']]
Iteration 3
[[ 0.   -1.   -1.5  -1.75]
 [-1.   -1.5  -1.75 -1.5 ]
 [-1.5  -1.75 -1.5  -1.  ]
 [-1.75 -1.5  -1.    0.  ]]
[['↑' '←' '←' '↓']
 ['↑' '↑' '↑' '↓']
 ['↑' '↑' '↓' '↓']
 ['↑' '→' '→' '↑']]
```

Questions 3.6 (5p)

Let's run policy iteration with $\gamma = 1$. Describe what is happening. Why is this the case? Give an example. What is γ trading off and how does it affect policy iteration? (max 8 sentences)

ANSWER HERE

The problem is that the convergence of the series is not guaranteed when $\gamma = 1$. Hence, running the code with $\gamma = 1$ the series diverges, since it never reaches the termination condition.

Since γ determines the present values of future rewards:

- if $\gamma = 0$, the agent only maximise the immediate rewards, reducing the return.
- if γ approaches 1, the agent takes more into account future rewards.

```
policy_iteration(world, gamma=1.0)
```