SAPIENZA
UNIVERSITÀ DI ROMA

Department of Computer, Control and Management
Engineering Antonio Ruberti

Assignment 1

Reinforcement Learning

Master program in Artificial Intelligence and Robotics

Professor:
Roberto Capobianco

TAs:
Andrea Fanti
Michela Proietti

Student:
Lavalle Leonardo 1838492

Academic Year 2023/2024

Contents

1 Theory

1.1

Value Iteration algorithm exploits the property of Bellman optimality equation of being a contraction mapping. The algorithm iteratively applies the contraction mapping (called T in literature) to the value function V until it converges to a *fixed point* V^* meaning that $V^* = TV^*$.
The value function obtained through this algorithm can be used to express a certain policy (not of our interest in this moment). But since we computed V^* in an iterative manner and not in closed form, we could wander how much is this result precise, or better, what is the quality of such policy?
Theory states that (for all states s):

$$V^{\pi_i}(s) \geq V^*(s) - \frac{2\gamma^i \|Q_0 - Q^*\|_\infty}{1-\gamma} \tag{1}$$

where $V^{\pi_i}(s)$ is the value function at iteration i, $V^*(s)$ is the optimal value function, γ is the discounted factor and $\|Q_0 - Q^*\|_\infty$ is the infinite norm between Q-function at $i = 0$ and the optimal one. The term after the minus in the right side equation can be interpreted as the error on the quality of the policy. In fact, an error on the quality of the policy that is at most ϵ can be represented in this way:

$$\frac{2\gamma^i \|Q_0 - Q^*\|_\infty}{1-\gamma} \leq \epsilon \tag{2}$$

How do we determine the minimum number of iterations i of Value Iteration that are needed if want such an error?
First of all, let's simplify it. Under the assumption that rewards are in $[0, 1]$ range, we can state that the maximum possible value of V is $\sum_{i=0}^{\infty} \gamma^i = \frac{1}{1-\gamma}$. Indeed, since both Q_0 and Q^* are in this $[0, \frac{1}{1-\gamma}]$ range, by the definition of the infinity norm, the maximum value is just $\frac{1}{1-\gamma}$. The formula becomes:

$$\frac{2\gamma^i}{(1-\gamma)^2} \leq \epsilon \tag{3}$$

From now on we consider this two assumptions: (i) discounted factor $\gamma \in [0, 1]$ for definition, but since $\gamma = 1$ makes Value function impossible to compute in practice, we have $\gamma \in [0, 1)$; (ii) being an error metric (e.g. $\epsilon = 0.001, \epsilon = 0.0002$), it's reasonable to consider $0 < \epsilon < 1$.
There are two ways of finding the minimum number of iterations i. The first one leads to the exact value and it exploits the fact that $\gamma \in [0, 1)$, while the second one is an overestimation of the latter. Both derivations start from equation 3. Let's derive the

first one:

$$\begin{aligned}
\gamma^i &\leq \frac{\epsilon(1-\gamma)^2}{2} \\
\log_\gamma \gamma^i &\leq \log_\gamma \frac{\epsilon(1-\gamma)^2}{2} \quad \text{(we apply } \log_\gamma \text{ to both sides)} \\
\log_\gamma \gamma^i &\geq \log_\gamma \frac{\epsilon(1-\gamma)^2}{2} \quad \text{(because } \gamma \in [0,1) \text{ and both sides are negative)} \\
i &\geq \log_\gamma \frac{\epsilon(1-\gamma)^2}{2} \quad \text{(for definition of logarithm)} \\
&= \frac{\ln \frac{\epsilon(1-\gamma)^2}{2}}{\ln \gamma} \quad (\log_a b = \frac{\ln b}{\ln a})
\end{aligned}$$

The minimum number of iterations i, according to the following equation, are:

$$i \geq \frac{\ln \frac{\epsilon(1-\gamma)^2}{2}}{\ln \gamma} \tag{4}$$

Let's derive the second one (always starting from 3):

$$\begin{aligned}
&\gamma^i \leq \frac{\epsilon(1-\gamma)^2}{2} \\
&\gamma^i \leq (1-(1-\gamma))^i \quad \text{(we add and substract 1 in the left side)} \\
&\gamma^i \leq (1-(1-\gamma))^i \leq e^{-(1-\gamma)i} \quad (1+x \leq e^x)
\end{aligned}$$

Starting from this overestimation:

$$\begin{aligned}
e^{-(1-\gamma)i} &\leq \frac{\epsilon(1-\gamma)^2}{2} \\
\ln e^{-(1-\gamma)i} &\leq \ln \frac{\epsilon(1-\gamma)^2}{2} \quad \text{(we apply ln on both sides)} \\
-(1-\gamma)i &\leq \ln \frac{\epsilon(1-\gamma)^2}{2} \\
(1-\gamma)i &\geq -\ln \frac{\epsilon(1-\gamma)^2}{2} \\
i &\geq \frac{-\ln \frac{\epsilon(1-\gamma)^2}{2}}{(1-\gamma)}
\end{aligned}$$

Indeed, this is the second equation which expresses the minimum number of iterations i:

$$i \geq \frac{-\ln \frac{\epsilon(1-\gamma)^2}{2}}{(1-\gamma)} \tag{5}$$

Both formula 4 and 5 depends on ϵ and γ. The equations are very similar and they only differ by the factor which multiplies $\ln \frac{\epsilon(1-\gamma)^2}{2}$: in 4 is multiplied by $\frac{1}{ln\gamma}$ and in 5 by $-\frac{1}{1-\gamma}$. By playing and tuning a bit these two parameters (within the ranges previously defined by the assumptions), it comes out that the *lower bound* for i coming from the first formula 4 is always lower than the one coming from the second one 5. We can

say that the first one is a more precise estimation. This simply happens because, in 5, we introduced an overestimation. As we could have expected, the more we increase the error ϵ the more the number of iterations decreased and the more we decreased the error ϵ the more the number of iterations increased. Intuitively, if we want a more accurate policy with a lower ϵ, we should compute more iterations of the algorithm.

1.2

We have to compute one iteration of the *Value Iteration* algorithm. The MDP under consideration has 7 states $\{s_i : i \in \{1..7\}\}$, with a reward function which gives 0.5 if we reach state s_1, 5 for the state s_7 and 0 for the other states. The rewards vector is $r = [0.5, 0, 0, 0, 0, 0, 5]$. The policy states that in every state, we perform action a_1. The transition probability of performing action a_1 in state s_6 is depicted in Figure 1.

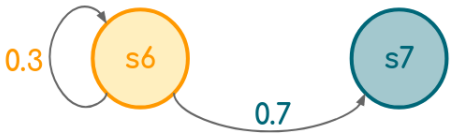

Figure 1

We recall that $V_{k+1}(s_i) = TV_k(s_i)$, where k is the iteration and s_i is the state with $i \in \{1...7\}$. Bellman optimality equation is:

$$TV_k(s_i) = \max_a (r(s,a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s,a)} V_k(s')) \tag{6}$$

Value function vector at time k is $V_k = [0.5, 0, 0, 0, 0, 0, 5]$ and $\gamma = 0.9$.
Indeed:

$$\begin{aligned}
V_{k+1}(s_6) &= TV_k(s_6) \\
&= \max_a (r(s_6,a) + \gamma \mathbb{E}_{s' \sim p(\cdot|s_6,a)} V_k(s')) \\
&= r(s_6,a_1) + \gamma \mathbb{E}_{s' \sim p(\cdot|s_6,a_1)} V_k(s') \quad \text{(there's only action } a_1\text{)} \\
&= r(s_6,a_1) + \gamma(0.3 \cdot V_k(s_6) + 0.7 \cdot V_k(s_7)) \\
&= 0 + 0.9 \cdot (0.3 \cdot 0 + 0.7 \cdot 5) \\
&= 0.9 \cdot (0.7 \cdot 5) \\
&= 0.9 \cdot 3.5 \\
&= 3.15
\end{aligned}$$

2 Code

2.1

The problem was to find an optimal policy for the FrozenLake environment with the *is_slippery* parameter set to *True*. This means that when the agent choose an action, it has $\frac{1}{3}$ probability of going in that direction and respectively $\frac{1}{3}$ and $\frac{1}{3}$ of moving in one of the perpendicular directions, because there's ice and the floor is slippery.
The environment has some *holes*, where the agent can go (they are *feasible* states) and consequently fall, resulting in an immediate episode termination. That's exactly the reason why the *policy_iteration.py* file was not correct. In the *policy evaluation* phase, the value function of these particular states shouldn't have been updated (because they are terminal states). In practice, at line 33 of the above mentioned file, the line should have been:

done = (state == end_state).all() or obstacles[state[0], state[1]] == 1

So the policy obtained without changing this line of code shouldn't be taken into consideration. In fact, after the launch of three sets of experiments (each one of 1000 episodes) the mean reward was respectively 0.244, 0.276 and 0.277. But again, we don't care about these results.

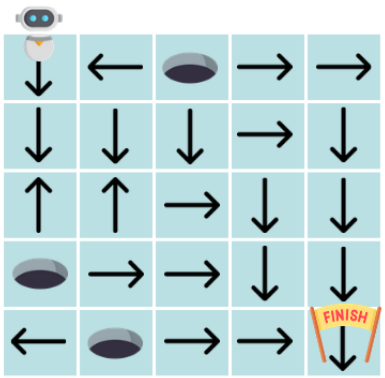

Figure 2

If we instead correctly change the line of code, we have a completely different scenario. After 1000 episodes the mean reward is 1 and this can be appreciated if we see the computed policy in Figure 2. All the actions of the neighboring states aim to stay away from the holes, and even in such a stochastic environment the agent succeed in safely reaching the goal.

2.2

This exercise was about applying the *iLQR* algorithm to the Pendulum environment. As it's known, *iLQR* is the right choice when dealing with non-linear dynamic systems. The formulas to make the optimal control happens are quite dense and difficult, but the algorithm is conceptually pretty straightforward. As the name suggests, the algorithm consists in performing iteratively *LQR*. The basic parameters to set are the time horizon for *LQR* (in our experiments is set to 20) and the number of iterations for *iLQR* (3 in our case).
After 100 episodes of the Pendulum environment the mean reward was -267.51. This value is not really significative and representative of the quality of the algorithm, since it strictly depends on the initial random position of the pendulum. The only way of being sure of the success or not, was to *render* the environment and see if the pendulum remained in the upright position (an *unstable equilibrium*) or not.