CS 4080-5080 - Reinforcement Learning
Homework 2 Report

Carlos E. L. P. X. Torres
University of Colorado at Colorado Springs
1420 Austin Bluffs Pkwy.
Colorado Springs, Colorado 80918
clopespi@uccs.edu

Abstract

Homework 2 Report for the class CS 4080-5080 - Reinforcement Learning due to 11/01/2021. On this assignment, it is implemented a reinforcement learning algorithm to teach an agent how to navigate in a 6x7 simplified parking lot environment from the entrance to any of the parking spots (P1 to P8). Two algorithms are used, Q-learning and Double Q-learning, and a comparison between them is made. Transfer learning is also discussed. A simulation is presented showing the implemented algorithms in action and also showing a chart of the learning progress.

1 Introduction

Homework 2 Report for the class CS 4080-5080 - Reinforcement Learning. On this assignment, it is implemented a reinforcement learning algorithm to teach an agent how to navigate in a 6x7 simplified parking lot environment from the entrance to any of the parking spots (P1 to P8). No reinforcement learning libraries are used, only some common libraries, like for numeric processing (numpy), math, random, and time.

In section 2, it is presented a model of the problem, describing the various components required for the agent and the environment (states, rewards, actions, barriers, edges avoiding handling).

In section 3, it is presented the two algorithms for learning the optimal policy using Q-learning and Double Q-learning. It is also presented how the latter improves the first.

In section 4, it is presented how the algorithms were implemented, and a simulation was created to illustrate it.

In section 5, it is discussed the learning process and transfer learning, reusing a previously created Q-table from one case in another case.

In conclusion, it is discussed the final considerations, problems faced, and how they were overcome during the work.

2 Problem Model

To model the 6x7 parking lot environment (Figure 1) as a reinforcement learning problem, we should formulate it in terms of a Markov Decision Process (MDP), consisting of a tuple $\{S, A, P, R, \gamma\}$ (Sutton and Barto 2018), where:

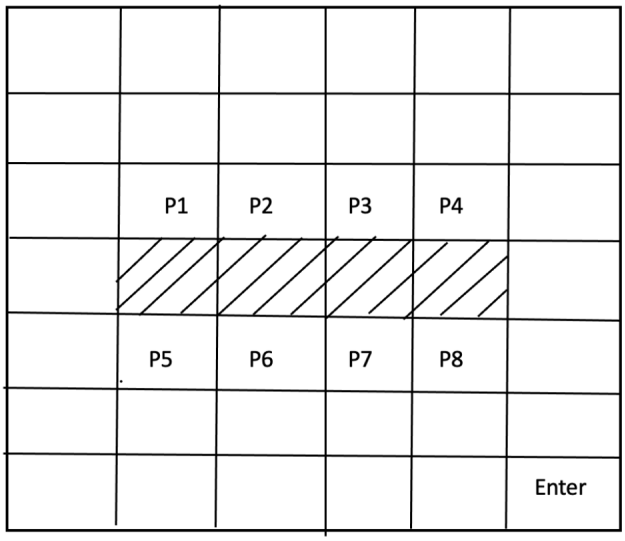

Figure 1: The 6x7 simplified parking lot environment

- S - The state space: a set of 42 states represented by a 6x7 2D matrix of cells (x, y) identifying each possible state inside the parking lot. The start state is represented as (5, 6) and the available parking spots states are P1 = (1, 2), P2 = (2, 2), P3 = (3, 2), P4 = (4, 2), P5 = (1, 4), P6 = (2, 4), P7 = (3, 4), P8 = (4, 4). But only one of the parking spots states are set as the target state each time.
- A - The action space: a set of 4 possible actions $\{N, S, E, W\}$ that the agent can take in each state inside the parking lot, each one being a directions (North, South, East, West). In this problem, not all actions are allowed in all states, due to the barrier states and the boundaries of the parking lot.
- P - The transaction probability function: defines how the agent can change states inside the parking lot. The function will serve to define the allowed actions inside each state (or cell) of the parking lot, associating probabilities to each state-action pair, and 0 (zero) for the ones that are not allowed. This will also serve to define the barriers inside the parking lot, by assigning 0 to an action that will lead to a barrier state. See Table 1 for the representation

of this function.

State / Action	N	S	E	W
s_1 (0,0)	0	0.5	0.5	0
s_2 (1,0)	0	0.33	0.33	0.33
s_3 (2,0)	0	0.33	0.33	0.33
s_4 (3,0)	0	0.33	0.33	0.33
s_5 (4,0)	0	0.33	0.33	0.33
s_6 (5,0)	0	0.5	0	0.5
s_7 (0,1)	0.33	0.33	0.33	0
s_8 (1,1)	0.25	0.25	0.25	0.25
s_9 (2,1)	0.25	0.25	0.25	0.25
s_{10} (3,1)	0.25	0.25	0.25	0.25
s_{11} (4,1)	0.25	0.25	0.25	0.25
s_{12} (5,1)	0.33	0.33	0	0.33
s_{13} (0,2)	0.33	0.33	0.33	0
s_{14} (1,2)	0	0	0	0
s_{15} (2,2)	0	0	0	0
s_{16} (3,2)	0	0	0	0
s_{17} (4,2)	0	0	0	0
s_{18} (5,2)	0.33	0.33	0	0.33
s_{19} (0,3)	0.5	0.5	0	0
s_{20} (1,3)	0	0	0	0
s_{21} (2,3)	0	0	0	0
s_{22} (3,3)	0	0	0	0
s_{23} (4,3)	0	0	0	0
s_{24} (5,3)	0.5	0.5	0	0
s_{25} (0,4)	0.33	0.33	0.33	0
s_{26} (1,4)	0	0	0	0
s_{27} (2,4)	0	0	0	0
s_{28} (3,4)	0	0	0	0
s_{29} (4,4)	0	0	0	0
s_{30} (5,4)	0.33	0.33	0	0.33
s_{31} (0,5)	0.33	0.33	0.33	0
s_{32} (1,5)	0.25	0.25	0.25	0.25
s_{33} (2,5)	0.25	0.25	0.25	0.25
s_{34} (3,5)	0.25	0.25	0.25	0.25
s_{35} (4,5)	0.25	0.25	0.25	0.25
s_{36} (5,5)	0.33	0.33	0	0.33
s_{37} (0,6)	0.5	0	0.33	0
s_{38} (1,6)	0.33	0	0.33	0.33
s_{39} (2,6)	0.33	0	0.33	0.33
s_{40} (3,6)	0.33	0	0.33	0.33
s_{41} (4,6)	0.33	0	0.33	0.33
s_{42} (5,6)	0.5	0	0	0.5

Table 1: Transition probability function representation

The barrier states themselves (s_{20}, s_{21}, s_{22}, s_{23}) will have a probability of 0 (zero), to keep the integrity of the function. Similarly, the parking spot states (s_{14}, s_{15}, s_{16}, s_{17}, s_{26}, s_{27}, s_{28}, s_{29}) will also be considered with probability 0 (zero) because they can assume target (or terminal) states, once achieved, the agent should not go anywhere else, and the episode ends.

- *R* - The reward signal: represented by a set of rewards from the environment to the agent. For this problem, two types of rewards will be used. For every step in the parking lot, the agent receives a reward of -0.1. And when the agent reaches the goal state (one of the parking spots that were selected as target), a reward of 1 is given. The negative rewards will make the agent prefer a policy with fewer steps to achieve the goal state.
- γ - The discount factor: can assume values $0 \leq \gamma \leq 1$. It increases the interest of the agent in having earlier rewards instead of future or later rewards. For this problem, it will be set initially to $\gamma = 0.99$, and can change during the learning process to see how it affects the algorithm.

3 Algorithms

Here is presented the two algorithms for learning the optimal policy using Q-learning (adapted from Sutton and Barto 2018) and Double Q-learning (adapted from Hasselt 2010).

Algorithm 1: Q-learning

1: **Parameters**:
   small $\epsilon > 0$
   $\gamma \in (0, 1]$
   step size $\alpha \in (0, 1]$
   maxNumberOfEpisodes > 0
2: **Initialize**, for all $s \in S, a \in A(s)$:
   $Q(s, a) \leftarrow 0$
3: **for** $i \leq$ numberOfEpisodes **do**
4: Initialize s
5: **for** step $\in$ episode **do**
6: Choose a from $Q(s, .)$ using the policy ϵ-greedy
7: Take action a, observe r, s', update $Q(s, a)$:
8: $Q(s, a) \leftarrow Q(s, a) + \alpha\big[r + \gamma max_a Q(s', a) - Q(s, a)\big]$
9: $s \leftarrow s'$
10: **if** s is terminal **then**
11: break
12: **end if**
13: **end for**
14: **end for**

Both algorithms are very similar to each other. They receive four parameters (line 1): ϵ, γ, step size α, and maxNumberOfEpisodes. The initial ϵ value that is being used is 0.8. The γ and α are set to 0.99. And the maxNumberOfEpisodes is set to 5000.

On line 2, for Q-learning, it starts by initializing the $Q(s, a) \leftarrow 0$ for all states $s \in S$ and $a \in A(s)$. For Double Q-learning, it initializes two Q tables, $Q^A(s, a) \leftarrow 0$ and $Q^B(s, a \leftarrow 0)$. That is the first different between the two algorithms.

On line 3, starts the main loop of the algorithms, using the maxNumberOfEpisodes parameter. In each iteration, it initializes a state s (Line 4) and starts the iterations through steps of each episode (Line 5).

On line 6, for Q-learning, it chooses an action a from $Q(s, .)$ using the ϵ-greedy policy. For Double Q-learning, it chooses the action a from both $Q^A(s, .)$ and $Q^B(s, .)$ together. That is the second different between the algorithms.

Algorithm 2: Double Q-learning

1: **Parameters**:
   small $\epsilon > 0$
   $\gamma \in (0, 1]$
   step size $\alpha \in (0, 1]$
   maxNumberOfEpisodes > 0
2: **Initialize**, for all $s \in S, a \in A(s)$:
   $Q^A(s, a) \leftarrow 0$, $Q^B(s, a) \leftarrow 0$
3: **for** $i \leq$ numberOfEpisodes **do**
4: Initialize s
5: **for** step $\in$ episode **do**
6: Choose a, based on $Q^A(s, .)$ and $Q^B(s, .)$ using the policy ϵ-greedy
7: Take action a, observe r, s'
8: Choose randomly to update either Q^A or Q^B
9: **if** UPDATE(Q^A) **then**
10: Define $a^* = argmax_a(Q^A(s', a))$
11: $Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(s', a)\big[r + \gamma Q^B(s', a^*) - Q^A(s, a)\big]$
12: **else**
13: Define $b^* = argmax_a(Q^B(s', a))$
14: $Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(s', a)\big[r + \gamma Q^A(s', b^*) - Q^B(s, a)\big]$
15: **end if**
16: $s \leftarrow s'$
17: **if** s is terminal **then**
18: break
19: **end if**
20: **end for**
21: **end for**

On line 7, the chosen action a is taken and it is observed r and s'. For Q-learning, it is also updated the $Q(s, a)$ value.

On line 8, for Double Q-learning, it is chosen randomly weather to update $Q^A(s, a)$ or $Q^B(s, a)$. That is the third difference between the algorithms.

On lines 10 to 14, for Double Q-learning, it is done the main difference of the algorithms. If Q^A is chosen, the action a^* is selected using $argmax_a(Q^A(s', a))$ and it is used to get the value $Q^B(s', a^*)$ to be used to update $Q^A(s, a)$. If Q^B is chosen the opposite process is performed, updating $Q^B(s', a)$ with the value from the b^* action taken from $argmax_a(Q^B(s', a))$. That is the double estimator idea presented by Hasselt.

After the updates, both algorithms end by updating the current state s with s', then checking if it is a terminal state to stop the steps loop.

3.1 Difference between both algorithms

As mentioned before, there are some key differences between the two algorithms chosen in this work. Q-learning is very powerful and efficient, but, in most cases, it overestimates the $Q(s, a)$ values by always choosing the maximum value, and its associated action. Whereas with Double Q-learning, it is mathematically proven by Hasselt that the expected value of Q^B for the action a^* is smaller or equal to the maximum value of $Q^A(s', a^*)$, i.e. $E(Q^B(s', a^*)) \leq MaxQ^A(s', a^*)$. This means that, if we perform large number of iterations, the expected value of $Q^B(s', a^*)$ is going to be smaller than the maximal value of $Q^A(s', a^*)$. In turn, $Q^A(s, a)$ is never updated with a maximum value and thus never overestimated.

The difference can be visualized in the chart of Figure 4, where the blue graph that represents Q-learning shows rewards almost always higher than the Double Q-learning one, in red. Meaning that the Double Q-learning, for the amount of episodes necessary to solve the problem, do not overestimate the Q values, not always picking the best action, and consequently, not always getting the higher rewards. This fact can look bad for the Double Q-learning algorithm, but, taking in consideration a large number of iterations, in the end it will have better results than Q-learning.

4 Implementation and Simulation

The algorithms were implemented in Python 3 using only native or common libraries like: random, time, sys, a numeric processing library called numpy, and a simulated environment, using a custom developed Open AI gym (Brockman et al. 2016) to model the simplified parking lot environment.

It was implemented a class named QLearning, that receives all the parameters for the algorithm, performs the training of the agent, and returns the generated data for the rewards and episodes created during the process for graph plotting purposes.

Another class named DoubleQLearning was created, inheriting from the QLearning class, and only changing the differences between the algorithms: the action picking and the two Q-tables updating processes.

The parameters used to train the agents for both algorithms can be viewed on Table 2 below. After running the training for both agents, it is generated a graph (Figure 4) with the results to compare the two algorithms.

Parameter	Value
Alpha (learning rate)	0.99
Alpha min	0.1
Gamma (discount factor)	0.99
Epsilon (exploration rate)	0.8
Epsilon min	0.001
Decay factor	0.042
Max Number of Steps/Episode	420
Max Number of Episodes	5000

Table 2: Learning parameters for the algorithms.

4.1 Policies Learned

After running the algorithm several times, the following policies were learned by the agent to take the car to each parking spot (P1 to P8). The parking spots are represented in Figure 2 and the policies for each one can be visualized in Figure 3.

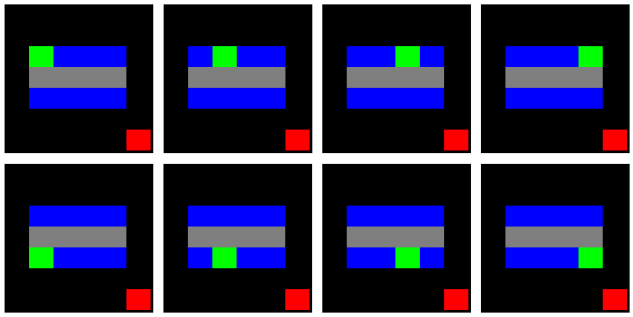

Figure 2: Parking lot spots representation in the simulation. From top left to bottom right, P1 to P8.

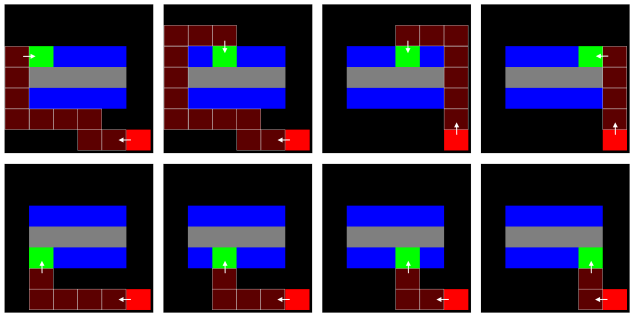

Figure 3: Policies learned for each spot represented in the simulation. From top left to bottom right, P1 to P8.

Policy for P1:
$\pi(s_{42}) = W \rightarrow \pi(s_{41}) = W \rightarrow \pi(s_{40}) = N \rightarrow \pi(s_{34}) = W \rightarrow \pi(s_{33}) = W \rightarrow \pi(s_{32}) = W \rightarrow \pi(s_{31}) = N \rightarrow \pi(s_{25}) = N \rightarrow \pi(s_{19}) = N \rightarrow \pi(s_{13}) = E \rightarrow \pi(s_{14})$

Policy for P2:
$\pi(s_{42}) = W \rightarrow \pi(s_{41}) = W \rightarrow \pi(s_{40}) = N \rightarrow \pi(s_{34}) = W \rightarrow \pi(s_{33}) = W \rightarrow \pi(s_{32}) = W \rightarrow \pi(s_{31}) = N \rightarrow \pi(s_{25}) = N \rightarrow \pi(s_{19}) = N \rightarrow \pi(s_{13}) = N \rightarrow \pi(s_7) = E \rightarrow \pi(s_8) = E \rightarrow \pi(s_9) = S \rightarrow \pi(s_{15})$

Policy for P3:
$\pi(s_{42}) = N \rightarrow \pi(s_{36}) = N \rightarrow \pi(s_{30}) = N \rightarrow \pi(s_{24}) = N \rightarrow \pi(s_{18}) = N \rightarrow \pi(s_{12}) = W \rightarrow \pi(s_{11}) = W \rightarrow \pi(s_{10}) = S \rightarrow \pi(s_{16})$

Policy for P4:
$\pi(s_{42}) = N \rightarrow \pi(s_{36}) = N \rightarrow \pi(s_{30}) = N \rightarrow \pi(s_{24}) = N \rightarrow \pi(s_{18}) = W \rightarrow \pi(s_{17})$

Policy for P5:
$\pi(s_{42}) = W \rightarrow \pi(s_{41}) = W \rightarrow \pi(s_{40}) = W \rightarrow \pi(s_{39}) = W \rightarrow \pi(s_{38}) = N \rightarrow \pi(s_{32}) = N \rightarrow \pi(s_{26})$

Policy for P6:
$\pi(s_{42}) = W \rightarrow \pi(s_{41}) = W \rightarrow \pi(s_{40}) = W \rightarrow \pi(s_{39}) = N \rightarrow \pi(s_{33}) = N \rightarrow \pi(s_{27})$

Policy for P7:
$\pi(s_{42}) = W \rightarrow \pi(s_{41}) = W \rightarrow \pi(s_{40}) = N \rightarrow \pi(s_{34}) = N \rightarrow \pi(s_{28})$

Policy for P8:
$\pi(s_{42}) = W \rightarrow \pi(s_{41}) = N \rightarrow \pi(s_{35}) = N \rightarrow \pi(s_{29})$

The algorithms were executed several times and the resulting policies are the best ones that were found. Notice that, for the policies P1 and P2, the algorithms found best to go to the left and up, whereas for P3 and P4, it went to the right and up.

Both algorithms converge after a few episodes, topping the rewards at around 20 episodes. The Double Q-learning one, generally converges a little later than Q-learning, for the reasons explained in section 3.1. The results can be viewed on Figure 4.

5 Transfer Learning Discussion

After each execution, the code saves the Q-table with the policy learned. In the case of Double Q-learning, it saves two tables Q^A and Q^B. Several tests were made using the saved tables as input to train the agent to learn a policy to a different parking spot, other than the one the table was saved. The results of these experiments are summarized on Table 3 below.

Parking Spots	Results
P1 $\leftrightarrow$ P2	50% fewer episodes to train
P3 $\leftrightarrow$ P4	50% fewer episodes to train
P5 $\leftrightarrow$ P8	50% fewer episodes to train
P1 & P2 $\leftrightarrow$ P3 & P4	Did not help significantly
P1 to P4 $\leftrightarrow$ P5 to P8	Did not help significantly

Table 3: Transfer learning results.

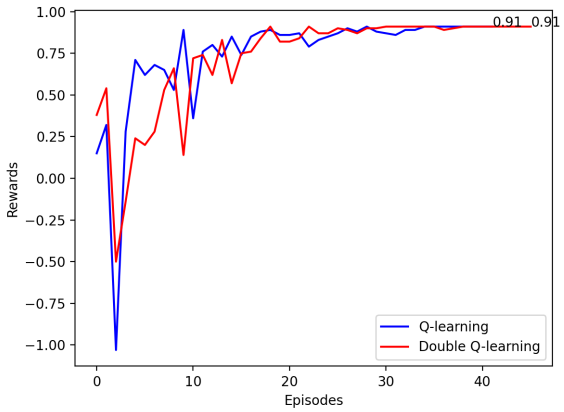

Figure 4: Cumulative rewards results for Q-learning and Double Q-learning.

For the bottom spots, any Q-table for the spots P5 to P8 helped decreased the number of episodes in average of 50% to train the agent for the other lower spots. Notice on Figure 3 that the resulting policies of those spots are very similar.

For the top spots, a different behavior occurred, where the top two left spots had good transfer learning results, along with the two top right ones. But between the left and right ones, there was no significant improvement. Notice that the policies for P1 and P2 are very similar, but quite different from the policies for P3 and P4.

6 Conclusion

The problem of the 6x7 simplified parking lot proposed on this work is a great example to use reinforcement learning to solve. It can be clearly formulated as an MDP and the initial policy created using the transition probability function. The algorithms selected to solve it were Q-learning and Double Q-learning, which showed a great performance and result.

The implementation of the algorithms was generally straightforward, with a few problems faced but overcame after some thought. As described in section 4, two classes were created for QLearning and DoubleQLearning, with the latter reusing most of the functionality of the first and only changing the functions related to action selection and Q-tables updates.

The algorithms main parameters are presented on Table 2. The decay factor is applied to the α (alpha) and ϵ (epsilon) parameters to dyamically decrease them after each episode.

References

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. *CoRR*, abs/1606.01540.

Hasselt, H. 2010. Double Q-learning. *Advances in neural information processing systems*, 23: 2613–2621.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.