Exercise Reinforcement learning

From the book Reinforcement learning, second ed. by Sutton and Barto, Example 5.1

The object of the popular casino card game of blackjack is to obtain cards the sum of whose numerical values is as great as possible without exceeding 21. All face cards count as 10, and an ace can count either as 1 or as 11. We consider the version in which each player competes independently against the dealer. The game begins with two cards dealt to both dealer and player. One of the dealer's cards is face up and the other is face down. If the player has 21 immediately (an ace and a 10-card), it is called a natural. He then wins unless the dealer also has a natural, in which case the game is a draw. If the player does not have a natural, then he can request additional cards, one by one (hits), until he either stops (sticks) or exceeds 21 (goes bust). If he goes bust, he loses; if he sticks, then it becomes the dealer's turn. The dealer hits or sticks according to a fixed strategy without choice: he sticks on any sum of 17 or greater, and hits otherwise. If the dealer goes bust, then the player wins; otherwise, the outcome—win, lose, or draw—is determined by whose final sum is closer to 21.

Playing blackjack is naturally formulated as an episodic finite MDP. Each game of blackjack is an episode. Rewards of +1, 1, and 0 are given for winning, losing, and drawing, respectively. All rewards within a game are zero, and we do not discount ($\gamma = 1$); therefore these terminal rewards are also the returns. The player's actions are to hit or to stick. The states depend on the player's cards and the dealer's showing card. We assume that cards are dealt from an infinite deck (i.e., with replacement) so that there is no advantage to keeping track of the cards already dealt. If the player holds an ace that he could count as 11 without going bust, then the ace is said to be usable. In this case it is always counted as 11 because counting it as 1 would make the sum 11 or less, in which case there is no decision to be made because, obviously, the player should always hit. Thus, the player makes decisions on the basis of three variables: his current sum

(12–21), the dealer's one showing card (ace–10), and whether or not he holds a usable ace. This makes for a total of 200 states.

Consider the policy that sticks if the player's sum is 20 or 21, and otherwise hits. To find the state-value function for this policy by a Monte Carlo approach, one simulates many blackjack games using the policy and averages the returns following each state. In this way, we obtained the estimates of the state-value function shown in Figure 1. The estimates for states with a usable ace are less certain and less regular because these states are less common. In any event, after 500,000 games the value function is very well approximated.

Exercise:

Please program the Monte Carle ϵ-greedy control algorithm to find the optimal policy.

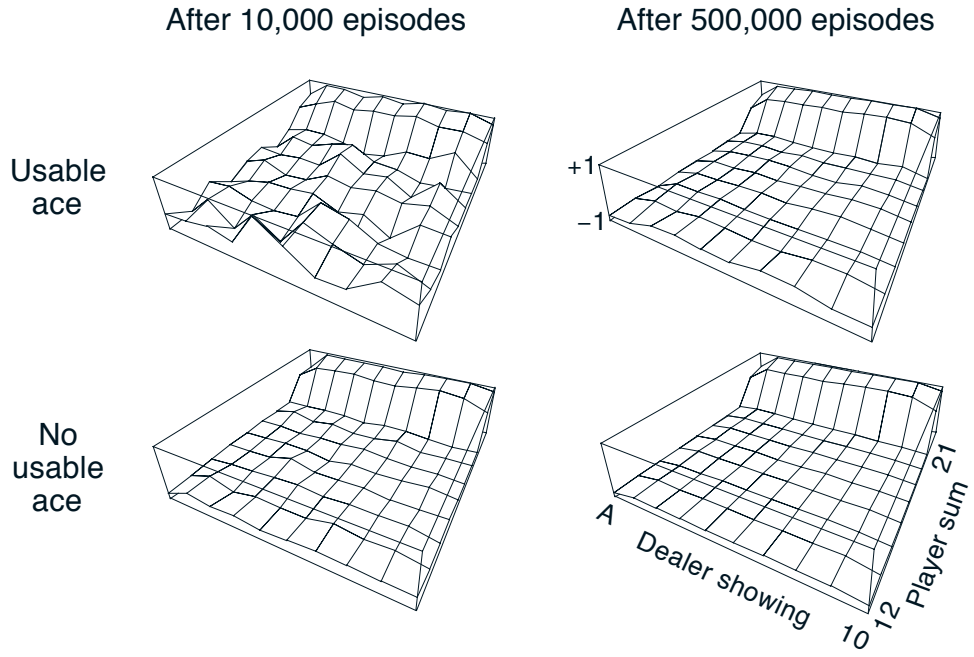

Figure 1: Approximate state-value functions for the blackjack policy that sticks only on 20 or 21, computed by Monte Carlo policy evaluation.

In the git https://github.com/volkerkrueger/edan95 I have provided several python files for jupyter notebook:

1. `blackjack.py`: a simulation environment for playing blackjack as explained above.

2. `Blackjack Playground.ipynb`: a playground with examples on how to use the blackjack simulation environment

3. `plotting.py`: plotting function for plotting the action value functions.

4. MC Prediction.ipynb: example solution for computing the plots in Figure 1. This file includes many comments to help understanding how the blackjack environment works.

5. `MC Control with Epsilon-Greedy Policies.ipynb`: starting point for your program: please fill in the missing parts:

    1. `make_epsilon_greedy_policy:` this is the where you give a ϵ/n probability to every possible action, and $1 - \epsilon + \epsilon/n$ probability for the best one.

    2. mc_control_epsilon_greedy: here, two steps are necessary.

        1. compute the new policy you want to follow using `make_epsilon_greedy_policy` from your present Q function

2. compute the new Q function based on a set of episodes sampled from the environment based on the new policy.

To run the above code you need to install openAI Gym.

Passing the Assignment

To pass the assignment, you need to a) prepare code that can compute the optimal policy and b) you need to discuss it in a little report.

Report

You will write a report of about two pages on your experiments:

1. You will describe the program you wrote and plot the graph showing your optimal policy. With how many episodes and how many policy improvement cycles was your result achieved?
2. Please read chapters 5-5.4 in the book “Reinforcement Learning, 2nd ed.” by Sutton and Barto and discuss the following questions
    1. Why do you have to use the Q function instead of the Value function? for MC control?
    2. Sect 5.3 talked about MC with exploring starts. We have not talked about exploring starts in the lecture. Why are exploring starts important, what is the problem with them and why can we omit them when using ϵ-greedy MC?
3. To write the report, you will use Overleaf.com. The submission procedure is described here: http://cs.lth.se/edan95/lab-programming-assignments/, but send me the Overleaf link only.

You must submit this report no later than one week after you have complete the lab.