Stellenbosch UNIVERSITY IYUNIVESITHI UNIVERSITEIT

Department of Mathematical Sciences
Division of Applied Mathematics

ML818 | Reinforcement Learning and Planning

Coursework 1

- Problems marked with * in the margin are for weeks 2 and 3.
- Solutions will be posted after week 3.

Q1. **(k-armed bandit problem)** A slot machine has 3 levers, numbered from 0 to 2, that returns a random reward R with distribution $\mathcal{N}(a, 1)$, where $a = 0, 1, 2$ is the lever number interpreted as an "action". Not knowing the underlying distribution of rewards, you "act" on the slot machine by pulling its levers many times until you find or "learn" the winning lever giving the largest expected reward

$$Q^* = \max_a Q(a) = \max_a \mathbb{E}[R|a].$$

(a) Write a program that "pulls" each lever in succession 1000 times and records the rewards obtained for each. Compare the histograms and means of the rewards obtained.

(b) Let $(A_i)_{i=1}^n$ be a sequence of actions (i.e., lever number pulled) and $(R_i)_{i=1}^n$ the corresponding sequence of rewards. Write a function that takes as input these two sequences and returns the estimated value function

$$Q_n(a) = \frac{\sum_{i=1}^n R_i \mathbb{1}_{A_i=a}}{\sum_{i=1}^n \mathbb{1}_{A_i=a}}$$

for each action $a = 0, 1, 2$. Here $\mathbb{1}_{A_i=a}$ is the indicator function equal to 1 if $A_i = a$ and 0 otherwise. Your function should have no nested loops and its output should be an array of 3 elements. Set $Q_n(a) = -\text{inf}$ for any action not present or seen in a trajectory.

(c) An ε-greedy policy for learning the winning lever proceeds as follows:

1- Initialise action list with $[A_1]$ and reward list with the corresponding reward $[R_1]$

2- Calculate $Q_1(a)$ using the function of Part (b)

3- With probability ε: Select next action A_2 randomly. With complementary probability $1 - \varepsilon$: Select next action as

$$A_2 = \arg\max_a Q_1(a)$$

4- Add A_2 and its corresponding reward R_2 in their respective list

5- Repeat Steps 2-4 with current lists N times.

Implement this algorithm. In the end, show the accumulated reward averaged over time as a function of time (i.e., iteration number) from $n = 1$ to $n = N = 1000$. Repeat for $\varepsilon = 0, 0.1, 0.3$, showing the results on the same plot. Re-run your code until you get a curve close to 2, the max expected reward. Explain your results.

(d) The value function $Q(a)$ can be estimated more efficiently using the following stochastic approximation:

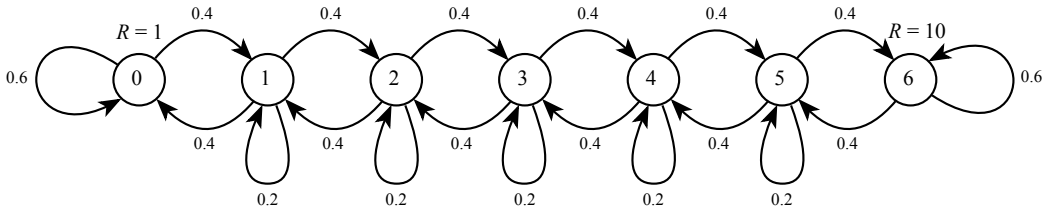
$$Q_n(a) = (1 - \alpha_n) Q_{n-1}(a) + \alpha_n R_n$$

for the realised action $A_n = a$ at time n (with corresponding reward R_n) and

$$Q_n(a') = Q_{n-1}(a')$$

for all other actions $a' \neq a$. Here $\alpha_n = 1/n$ is the annealing sequence. Implement this iteration and plot as in Part (c) the accumulated reward averaged over time for $\varepsilon = 0, 0.1, 0.3$. Use $Q_0(a) = 0$ for the initial value in the iteration. Re-run your code until you get a curve close to 2, the max expected reward. Explain your results.

Q2. (Linear model: Markov reward process) We consider the simple linear model with 7 states, discussed in class. The reward is a function $r(s)$ of the current state $S_t = s \in \{0, 2, \ldots, 6\}$, taken to be $r(0) = 1$, $r(6) = 10$, and 0 otherwise. Thus R_{t+1} is deterministic given $S_t = s$, and is equal to 1 when leaving state 0 and 10 when leaving state 6.



(a) For the transition probabilities $p(s'|s)$ shown above, calculate the value function $v(s)$ by directly solving the linear Bellman equation. Show the result as a plot for the values for $s = 0, 1, \ldots, 6$. Repeat the calculation for $\gamma = 0, 0.1, 0.2, \ldots, 0.9$ and show the results on the same plot.

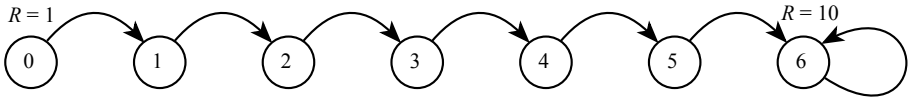
(b) Calculate $v(s)$ by solving the linear Bellman equation iteratively. Show the results as in Part (a). [*]

(c) Write a program that generates one trajectory of the Markov reward process from time $t = 1$ to $t = N$ and the return accumulated over that time. Pay attention to the extremal states 0 and 6.

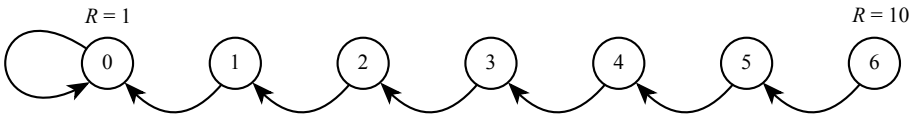
(d) Estimate $v(2)$ for $\gamma = 0.5$ using many simulated trajectories. Try different sample sizes (numbers of trajectories) and final times to make sure your results converge. Compare with the result of Part (a).

Q3. (Linear model: Markov decision process) We modify the model of Q2 to make it a decision process by including two types of deterministic actions for each state:

- MoveRight action ($a = 1$): Move from state $S_t = i$ to $S_{t+1} = i + 1$, unless $S_t = 6$ in which case stay there so $S_{t+1} = 6$:



- MoveLeft action ($a = 0$): Move from state $S_t = i$ to $S_{t+1} = i - 1$ unless $S_t = 0$ in which case stay there so $S_{t+1} = 0$:



(a) Write down the transition probabilities $p(s'|s, a)$ for $a = 0$ and $a = 1$ as two 7×7 matrices.

(b) Calculate the value function $v_\pi(s)$ for the full MoveRight policy, $\pi(s) = 1$ for all s, by solving the linear Bellman equation directly. Show the result in a plot as in Q2(a). Repeat for the full MoveLeft policy, $\pi(s) = 0$ for all s, and then for the random $\frac{1}{2}$MoveLeft + $\frac{1}{2}$MoveRight policy. Compare and explain your results.

(c) Write down the Bellman optimality equation for $v_*(s)$ and simplify it knowing that there are only two actions to compare and only one s' for a given s when s is not an extremal state (0 or 6). What is the simplified equation for the extremal states? You should end up with a max over two expressions for $a = 0, 1$. [*]

(d) Use your simplified Bellman optimality equation in an iterative way to find the optimal value function $v_*(s)$ for $\gamma = 0.5$ and the corresponding optimal (deterministic) policy $\pi_*(s)$, represented as $0, 1$. The optimal policy is obtained iteratively as the solution of the max at each iteration. Use synchronous updating in the iteration by copying v_k into a temporary vector. Show on the same plot $v_*(s)$ and the $v(s)$ obtained before for the fixed MoveLeft and MoveRight policies. Explain your results. [*]

(e) (Optional) Repeat Part (d) using in-place or asynchronous updating without the temporary copy of v_k. [*]

(f) Suppose you add a third action, called Stay, such that $S_{t+1} = S_t$ for all states. Explain why adding this action does not change the optimal value function and optimal policy. Would the optimal solution change if moves from state 3 to state 6 were added? [*]

Q4. (Linear model: Temporal difference) [*]

(a) Implement the TD(0) algorithm for finding the value function $v_\pi(s)$ associated with the random $\frac{1}{2}$MoveLeft + $\frac{1}{2}$MoveRight policy for the linear model. Use $\gamma = 0.5$, $\alpha = 0.1$, 1000 episodes, and $N = 1000$ time steps per episode. Only show the final estimates, comparing the estimated v_π with the result found in Q3 on the same plot.

(b) Implement the Sarsa algorithm for finding q_* and π_* for the linear model. Use $\gamma = 0.5$, $\alpha = 0.2$, $\varepsilon = 0.1$, 1000 episodes, and $N = 1000$ time steps. Only show the final estimates.

(c) Implement the Q-learning algorithm for finding q_* and π_* as in Part (b).

(d) Explain the difference between Sarsa and Q-learning.

Q5. (Gridworld) Use the Q-learning algorithm to find the optimal value function and policy of the gridworld model found in Sutton & Barto, p. 60. Compare with the optimal solution obtained for $\gamma = 0.9$ (with $\varepsilon = 0.1$ and $\alpha = 0.2$) shown on p. 65. [*]

Coding tips

- Don't define functions to answer a question, e.g., def Q1(bla, bla, bla).
- Never write a function that returns a plot.
- Don't define functions if you don't intend to use them more than 3 times.
- Don't re-use bits of code if you can use a loop instead.
- Clean your code: no unnecessary spaces, variables, containers, while True loops, etc.
- Code briefly but clearly. Don't over-write code and avoid being pythonic.
- Organise your notebook: use sections, subsections, etc., and put a header with your name.