Garrett Partenza
Advanced Reinforcement Learning
February 18, 2022

Exercise 1

Question 1. (a) Give an equation for v_* in terms of q_*. Give an equation for q_* in terms of v_* and the four-argument p. (c) Give an equation for π_* in terms of q_*. (d) Give an equation for π_* in terms of v_* and the four-argument p. (e) Rewrite the four Bellman equations for the four value functions in terms of the three- argument function p (Equation 3.4) and the two-argument function r (Equation 3.5).

(a)

$$v_*(s) = argmax\ q_*(s,a) \tag{1}$$

(b)

$$q_*(s,a) = argmax\ p(s',r|s,a)(r + v_*(s') \tag{2}$$

(c)

$$\pi_*(a|s) = argmax\ q_*(s,a) \tag{3}$$

(d)

$$\pi_*(a|s) = argmax \sum_{s',r \in S} p(s',r|s,a)(r + v_*(s') \tag{4}$$

(e)

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)\Big(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)v_\pi(s')\Big) \tag{5}$$

$$v_*(s) = argmax\ \Big(r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)v_\pi(s')\Big) \tag{6}$$

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)\Big(\sum_{a \in A} \pi(a'|s')q_\pi(s',a')\Big) \tag{7}$$

$$q_*(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)\Big(argmax\ q_\pi(s',a')\Big) \tag{8}$$

Question 2. (a) The policy iteration algorithm on page 80 has a subtle bug in that it may never terminate if the policy continually switches between two or more policies that are equally good. This is okay for pedagogy, but not for actual use. Modify the pseudocode so that convergence is guaranteed. (b) Is there an analogous bug in value iteration? If so, provide a fix; otherwise, explain why such a bug does not exist.

(a) To correct this bug, we can add an if-statement before we assign the updated policy $\pi(s)$ to the argmax action of that state, such that it only assigns the new argmax policy if that state-action value is strictly greater than the current policy.

(b) There is not an analogous bug in value iteration because there is not a similar state-action policy improvement step. Instead of doing two steps (evaluation and improvement), value iteration updates state-values until delta converges. Thus even if two actions produce the same state value, delta would still converge since the difference between the old and new policies would be zero.

Question 3. (a) How would policy iteration be defined for action values? Give a complete algorithm for computing q_*, analogous to that on page 80 for computing v_*. Please pay special attention to this exercise, because the ideas involved will be used throughout the rest of the book. (b) What is the analog of the value iteration update Equation 4.10 for action values, $q_{k+1}(s, a)$?

(a) (1) Initialization
  (i) $Q(s) \leftarrow$ random value $\in R$.
  (ii) $\pi(s) \leftarrow$ random policy
 (2) Policy Evaluation
  (i) $\triangle \leftarrow 0$
  (ii) For all states $\in S$
  $q \leftarrow q(s, a)$
  $q(s, a) \leftarrow \sum_{s' \in S} p(s'|s, a)(r(s'|s, a) + \gamma q(s', a'))$
  $\triangle \leftarrow max(\triangle, q - q(s, a))$
  continue until convergence
 (3) Policy Improvement
  (i) stable = true
  (ii) For all states $\in S$
  old action $\leftarrow \pi(s)$
  $\pi(s) \leftarrow q(s, a) \leftarrow argmax \sum_{s' \in S} p(s'|s, a)(r(s'|s, a) + \gamma q(s', a'))$
  if old and new actions differ, set stable to false
  (iii) *if stable, halt*

(b) (1) Loop
  (i) For all states $\in S$
  old action $\leftarrow \pi(s)$
  $q(s, a) \leftarrow q(s, a) \leftarrow argmax \sum_{s' \in S} p(s'|s, a)(r(s'|s, a) + \gamma q(s', a'))$
  $\triangle \leftarrow max(\triangle, q - q(s, a))$
  loop until convergence

Question 4. (a) What can be determined qualitatively about the optimal policy in states x and y (i.e., just by looking at the transition and reward structure, without running value/policy iteration to solve the MDP)? (b) Apply policy iteration, showing each step in full, to determine the optimal policy and the values of states x and y. Assume that the initial policy has action c in both states. (c) What happens to policy iteration if the initial policy has action b in both states? Does discounting help? Does the optimal policy depend on the discount factor (in this particular MDP)?

(a) The agent wants to arrive to the terminal state z as fast as possible. This is because in all other states, moves result in a negative reward and force the agent to factor in completion time. Furthermore, state y results in a higher negative reward than state x, and as a result, it may be more beneficial to move to state x rather than directly to state z since the probability of noise is much higher moving directly from y to z.

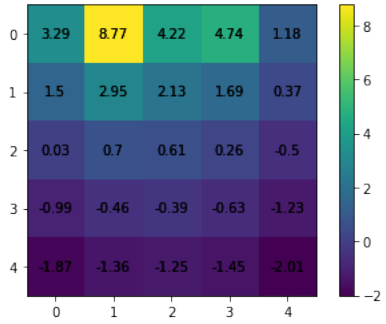
(b) (1) Initialize
  (i) $\pi(x) = c, \pi(y) = c$
 (2) Evaluate
  (i) $v(x) = -1 + 0.1v(z) + 0.9v(x) = -10$
  (ii) $v(y) = -2 + 0.1v(z) + 0.9v(y) = -20$
 (3) Update
  (i) $\pi_{old}(x) = c, \pi_{old}(y) = c$
  (ii) $\pi_{new}(x) = argmax_a(q(x, b), q(x, c)), \pi_{new}(y) = argmax_a(q(y, b), q(y, c))$
  (iii) $q(x, b) = -17, q(x, c) = -10, q(y, b) = -13, q(x, c) = -19$
  (iv) $\pi_{new}(x) = c, \pi_{new}(y) = b$
  (v) *not stable*

(4) Evaluate
  (i) $v(x) = -1 + 0.1v(z) + 0.9v(x) = -10$
  (ii) $v(y) = -2 + 0.8v(z) + 0.2v(y) = -12.5$
(5) Update
  (i) $\pi_{old}(x) = c, \pi_{old}(y) = b$
  (ii) $\pi_{new}(x) = argmax_a(q(x,b), q(x,c)), \pi_{new}(y) = argmax_a(q(y,b), q(y,c))$
  (iii) $q(x,b) = -13, q(x,c) = -10, q(y,b) = -12.5, q(x,c) = -13.5$
  (iv) $\pi_{new}(x) = c, \pi_{new}(y) = b$
  (v) *stable*

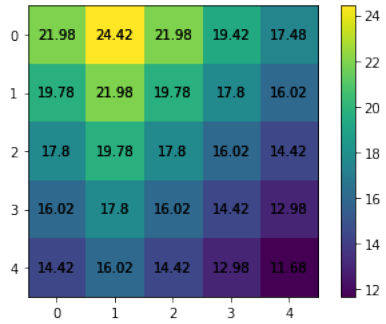
(c) If the initial policy has action b in both states, then we result in a unsolvable system of linear equations. Discounting would help with this issue, however, its value would effect our policy. For example, if the discount is very small, the agent may not care about the long term effect of staying in state y as it attempts to move towards the terminal state, since the downstream penalty is so heavily nulled by the small discount factor.

Question 5. Implementing dynamic programming algorithms.

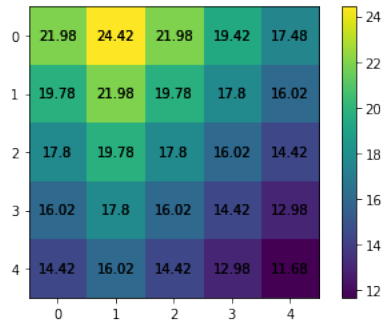
(a) The state value function found by a random policy is



(b) The optimal policy found by value iteration is (0, 0): 'down', (0, 1): 'right', (0, 2): 'up', (0, 3): 'left', (0, 4): 'up', (1, 0): 'up', (1, 1): 'left', (1, 2): 'left', (1, 3): 'down', (1, 4): 'right', (2, 0): 'down', (2, 1): 'left', (2, 2): 'up', (2, 3): 'left', (2, 4): 'down', (3, 0): 'up', (3, 1): 'down', (3, 2): 'down', (3, 3): 'left', (3, 4): 'right', (4, 0): 'down', (4, 1): 'left', (4, 2): 'right', (4, 3): 'down', (4, 4): 'left'. The state value function found by policy iteration is



(c) The optimal policy found by value iteration is (0, 0): 'right', (0, 1): 'up', (0, 2): 'left', (0, 3): 'down', (0, 4): 'left', (1, 0): 'right', (1, 1): 'up', (1, 2): 'up', (1, 3): 'left', (1, 4): 'left', (2, 0): 'right', (2, 1): 'up', (2, 2): 'up', (2, 3): 'up', (2, 4): 'up', (3, 0): 'right', (3, 1): 'up', (3, 2): 'up', (3, 3): 'up', (3, 4): 'up', (4, 0): 'right', (4, 1): 'up', (4, 2): 'up', (4, 3): 'up', (4, 4): 'up'. The state value function found by policy iteration is



Question 6. Jack's car rental problem.

(a)

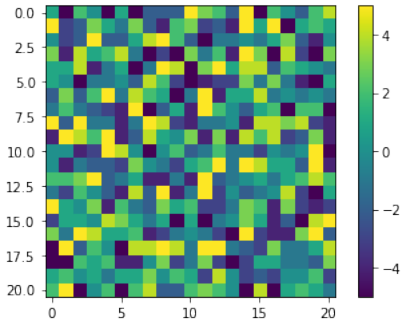


FIGURE 1. (a) Initialization

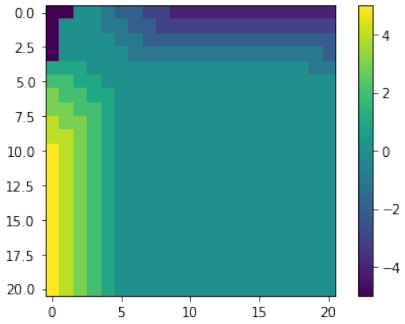


FIGURE 2. (a) Step 3

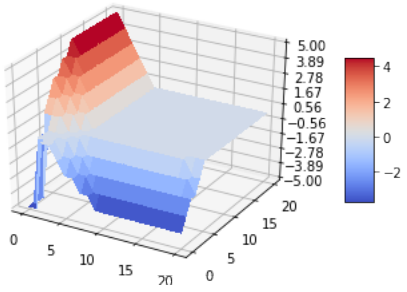


FIGURE 3. (a) Landscape

(b) For the first change I will add 2 reward whenever the number of cars moved from the second location to the first is greater than zero. For the second change, I will add -4 reward for each state which is greater than 10 cars. Looking at the difference in policies between part (a) and part (b), we see

that the model is more willing to bring cars from location two to location one (you can see this by the second policy having a darker shade in the top left corner, indicating a more negative number and thus more cars transferred from location two to location one). This makes sense, considering that in part (b) we allowed one free car for moves from location two to location one, and our model exploits this difference.

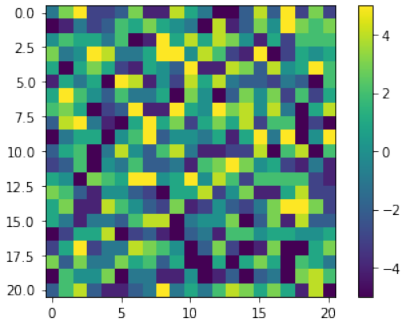


FIGURE 4. (b) Initialization

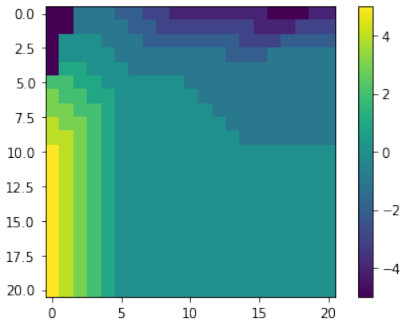


FIGURE 5. (b) Step 4

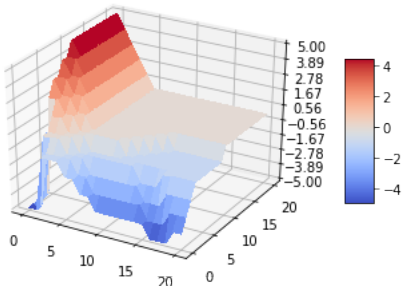


FIGURE 6. (b) Landscape