EL2805 Reinforcement Learning

Exercise Session 4

November 26, 2019

Department of Automatic Control
School of Electrical Engineering
KTH Royal Institute of Technology

4 Exercises

Some of these exercises have been inspired by, or taken from, [1]. If you want to solve more exercises, see any of those books.

4.1

In, for example, the game of tic-tac-toe, many positions appear different but are really the same because of symmetries.

a) How can we take this into account when we define the state-space? Do you think the learning process can benefit from this?

b) Suppose an opponent does not take symmetries into account (that is, his action-selection policy might be different in symmetrically equivalent states). Is it still a good idea to do what you proposed in a)?

4.2

Explain in your own words what the difference between an *on-policy* and *off-policy* learning algorithm is. What type is Q-learning, and why?

4.3

In ε-greedy action selection, for the case of two actions and $\varepsilon = 0.5$, what is the probability that the greedy action is selected (assuming it is unique)? What if we have three actions and $\varepsilon = 1/5$?

4.4

You start in the square marked *Start* in the gridworld of Figure 1. For every transition, you receive a reward -1, except if you step into:

- a *Hole*, in which case you receive -100;
- the *Finish* square, in which case you receive a reward +10.

The task is episodic and reset whenever you step into a *Hole* or the *Finish* square.

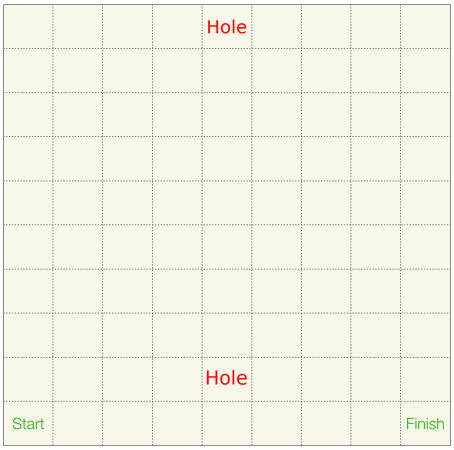

Figure 1: The gridworld for Problem 4.6.

a) Explain in your own words the difference between the SARSA and Q-learning algorithms.

b) Draw (in Figure 1) a possible optimal path for the state-action function computed using Q-learning and SARSA, respectively, with ε-greedy action selection, $\varepsilon = 0.6$. Motivate.

4.5

Suppose action selection is greedy (with respect to the current estimate of the state-value function). Is Q-learning then exactly the same algorithm as SARSA? Will they make exactly the same action selections and updates to their state-action functions?

4.6

Consider a system in three possible states that can be controlled using one of two possible actions in each state. We run the Q-learning algorithm with the initial Q-values equal to zero (see Table 1). At each step we observe the current state, receive some reward, observe a selected action and the resulting next state. These observations for the 5 first steps are presented in Table 2.

a) Provide the updated Q-values after these steps, assuming that the discount factor is $\lambda = 0.8$ and the learning rate is fixed to $\alpha = 0.5$.

Q	S_1	S_2	S_3
a_1	0	0	0
a_2	0	0	0

Table 1: Initial Q-table.

b) What would be the optimal policy if the Q-learning algorithm had converged after these 5 steps?

Time	Current State	Reward	Action	Subsequent State
1	S_1	-2	a_1	S_3
2	S_3	6	a_1	S_3
3	S_3	4	a_2	S_2
4	S_2	-2	a_1	S_2
5	S_2	2	a_2	S_1

Table 2: Evolution of system trajectory.

4.7

Consider the system in Figure 2, which consists of four states $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$ and two actions $\mathcal{A} = \{A, B\}$. As can be seen from the figure, the transitions and rewards are deterministic – for example, $p(s' = s_1 | s = s_4, a = A) = 1$ and $r(s = s_4, a = A) = 70$. All of this (except for $\mathcal{S}$ and $\mathcal{A}$) is initially unknown to us, and we aim to learn how to optimally control this system.

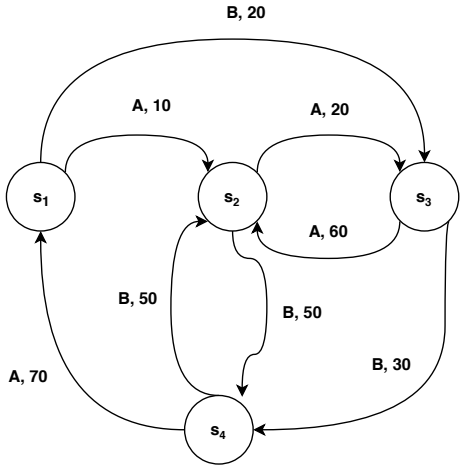

Figure 2: The system we aim to learn.

In order to do so, we employ the Q-learning algorithm with $\lambda = 0.9$ and $\alpha = 1$. Assume that we start in state s_1 and apply the sequence $\{A, A, B, A, B, A\}$ of actions. What is our state-action function? Assume we initialized it with zeros. What greedy policy does this state-action function correspond to?

5 Solutions

Solution to Problem 4.1

Part a)

All symmetrically equivalent states can be represented by one single state in the state-space. The number of states for which we need to estimate a state-action value will then be (potentially) much smaller (recall, $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$). For a given set of samples (trajectories), we are expected to be able to compute better estimates (since we have more samples available per state-action pair).

Part b)

Probably not: We will expect (by assumption) the same behaviour from the opponent in the symmetrically equivalent states.

Solution to Problem 4.2

On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data. Usually, in on-policy methods, the agent commits to always exploring and tries to find the best policy that still explores.

Q-learning is off-policy since the learned action-value function directly approximates q^* (the optimal action-value function), independently of the policy followed.

Solution to Problem 4.3

There are two ways in which the greedy action might be selected: *i)* with probability $1 - \varepsilon = 1/2$ we select to exploit, which means we take the greedy action, and *ii)* with probability $\varepsilon = 1/2$ we choose to explore. In this case, there is a subsequent probability $1/2$ that the greedy action is selected (we chose uniformly between the two available actions). In total, the probability is then $1/2 + 1/4 = 3/4$.

For three actions and $\varepsilon = 1/5$, we have probability $4/5 + 1/5 \times 1/3 = 13/15$ of selecting the greedy action.

Solution to Problem 4.4

Part a)

The difference is in the update of the action-state values. Q-learning *assumes* that the action taken from the next state is the greedy one;

$$q(s,a) \leftarrow q(s,a) + \alpha \left(r(s,a) + \lambda \max_{a'} q(s',a') - q(s,a) \right),$$

whereas SARSA updates its action-state values based on the *actual* action taken:

$$q(s,a) \leftarrow q(s,a) + \alpha \left(r(s,a) + \lambda q(s',a') - q(s,a) \right).$$

Note that the *actual* subsequent action taken in Q-learning is *not* necessarily the greedy one!

Part b)

Q-learning learns the optimal path (the shortest path). SARSA takes into account that when implementing our policy, we use ε-greedy exploration and risk falling down a hole. It finds a safer (but longer) path.

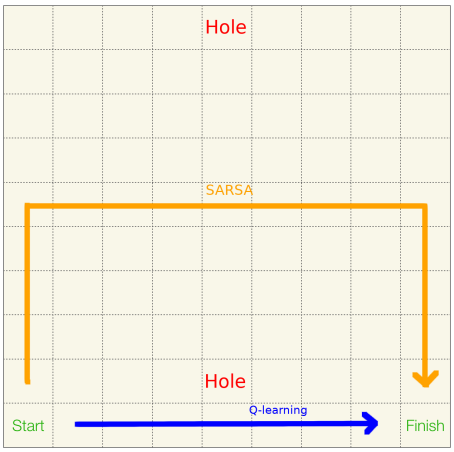

Figure 3: The gridworld for Problem 4.6.

Solution to Problem 4.5

No. The action that is optimal according to the current state-action function in Q-learning is *not* necessarily the best action after it has been updated. Hence, we will potentially implement another action than what we used in the update (in contrast to SARSA).

In particular, consider the Q-learning update:

$$q^{(k+1)}(s,a) = q^{(k)}(s,a) + \alpha\left(r(s,a) + \lambda \max_{a'} q^{(k)}(s',a') - q^{(k)}(s,a)\right).$$

The state-action function is updated as if the action $\text{argmax}_{a'}\, q^{(k)}(s',a')$ will be applied in the next time-step. However, the action that will *actually* be applied is $\text{argmax}_{a'}\, q^{(k+1)}(s',a')$. Note that, in general,

$$\underset{a'}{\text{argmax}}\left(q^{(k)}(s',a')\right) \neq \underset{a'}{\text{argmax}}\left(q^{(k+1)}(s',a')\right).$$

On the other hand, for SARSA, we decide upon an action for the next time-step before updating the state-action function:

$$a' = \underset{a}{\text{argmax}}\, q^{(k)}(s',a),$$

and then update

$$q^{(k+1)}(s,a) = q^{(k)}(s,a) + \alpha\left(r(s,a) + \lambda q^{(k)}(s',a') - q^{(k)}(s,a)\right).$$

The action a' we decided on is what we will also *actually* implement.

Solution to Problem 4.6

Part a)

$$Q(S,a) = Q(S,a) + \alpha\left[R(S) + \lambda \max_{a'}\{Q(S',a')\} - Q(S,a)\right]$$

Step 1: $S = S_1, R(S_1) = -2, a = a_1, S' = S_3$ Hence,

$$Q(S_1,a_1) = Q(S_1,a_1) + \alpha\left[R(S_1) + \lambda \max_{a'}\{Q(S_3,a')\} - Q(S_1,a_1)\right] = -1$$

Step 2: $S = S_3, R(S_3) = 6, a = a_1, S' = S_3$

$$Q(S_3,a_1) = Q(S_3,a_1) + \alpha\left[R(S_3) + \lambda \max_{a'}\{Q(S_3,a')\} - Q(S_3,a_1)\right] = 3$$

Q	S_1	S_2	S_3
a_1	-1	0	0
a_2	0	0	0

Q	S_1	S_2	S_3
a_1	-1	0	3
a_2	0	0	0

Step 3: $S = S_3, R(S_3) = 4, a = a_2, S' = S_2$

$$Q(S_3, a_2) = Q(S_3, a_2) + \alpha \left[R(S_3) + \lambda \max_{a'} \{Q(S_2, a')\} - Q(S_3, a_2) \right] = 2$$

Q	S_1	S_2	S_3
a_1	-1	0	3
a_2	0	0	2

Step 4: $S = S_2, R(S_2) = -2, a = a_1, S' = S_2$

$$Q(S_2, a_1) = Q(S_2, a_1) + \alpha \left[R(S_3) + \lambda \max_{a'} \{Q(S_2, a')\} - Q(S_2, a_1) \right] = -1$$

Q	S_1	S_2	S_3
a_1	-1	-1	3
a_2	0	0	2

Step 5: $S = S_2, R(S_2) = 2, a = a_2, S' = S_1$

$$Q(S_2, a_2) = Q(S_2, a_2) + \alpha \left[R(S_3) + \lambda \max_{a'} \{Q(S_1, a')\} - Q(S_2, a_2) \right] = 1$$

Q	S_1	S_2	S_3
a_1	-1	-1	3
a_2	0	1	2

Part b)

The optimal policy: $\pi(S_1) = a_2, \pi(S_2) = a_2, \pi(S_3) = a_1$.

Solution to Problem 4.7

$q(s, a)$:

	A	B
s_1	10	47
s_2	20	0
s_3	78	30
s_4	79	0

which corresponds to the policy $a(s_1) = B, a(s_2) = A, a(s_3) = A, a(s_4) = A$.

References

[1] R. S. Sutton, A. G. Barto, F. Bach, *et al.*, *Reinforcement learning: An introduction*. MIT press, 1998.

[2] D. Bertsekas, *Dynamic programming and optimal control*, vol. 1. Athena Scientific, 1995.