

Solutions to the exercises in 'Reinforcement learning' by Sutton and Barto

Remy Messadene

January 2020

Here are the solutions to all the exercises featured in what many people would call the *bible of Reinforcement learning*. Please reach out to me if you find any mistake in these notes or an interesting alternative solution to one of the questions: I would be happy to hear your suggestion, ideas, or comments! Have fun!

1 Introduction

1.1 Self-play

Solution: When playing against a random opponent, the agent is led to explore the state space faster whereas playing against itself already brings a certain incentive to exploit the information it has. Playing against itself will therefore make the exploration slower.

In term of the final optimal policy that is gonna be learned, this depends on the variation of the ϵ of the opponent agent with time. If it monotonically decreases to zero, it is possible that the optimal policies will not be the same as the agent will converge to a local optimal policy. On the other side, for $\epsilon \neq 0$ and for a large number of games, our agent might be able to properly explore the space and converge to an optimal policy.

1.2 Symmetries

Solution: Conceptually, one would be interested in considering the equivalence classes of the state space with respect to rotations by $\frac{\pi}{2}k$, $k \in \{0, 1, 2, 3\}$ as well as horizontal and vertical reflexion with respect to their respective axis of symmetry.

This would improve the learning process by reducing space and time complexities of the algorithm.

Regarding the case where the opponent does not take advantage of symmetries and has a certain bias toward a certain configuration or lack of experience in their equivalent ones, then it would be beneficial for our agent to exploit

this by not doing the above identification and reduction of the state space as different equivalent states will lead to different values.

1.3 Greedy Play

Solution: The learning will be inexistent ($\epsilon = 0$). Even if it has had a sufficient training prior to becoming fully greedy, his strategy will likely be suboptimal. Moreover, while it could win a lot of games at the start against another learning agent, the latter will learn our agent's policy and start beating him consistently on the long run.

1.4 Learning from exploration

Solution: Learning after each moves implies that one is given a reward function measuring the goodness of the move. Assuming it is well calibrated, then learning from exploratory moves will make the discovery of the optimal policy more likely than a purely greedy one, which might be stuck in a sub-optimal strategy which is characterised by the first random moves than have been taken at the start of the learning process. On the long term, the agent who learnt from exploratory moves will consistently beat the one who did not, assuming both of them continue to make exploratory moves at the same frequency.

1.5 Other improvements

Solution: Regarding improvement to the current structure, there are several ways the agent be improved: I will suggest two. First, in the exploratory regime, instead of considering a random move, one could weight the other actions inversely proportionally to the number of time it selected it in the past to force a quicker overall exploration of the state space (this works in this case as the state space is not too big). Second, one could also implement pre-terminal positions that an experienced player with the game of tic-tac-toe is familiar with as well as their symmetries within the agent and set a hard-value for the terminal nodes in the tree associated to them (depending on whether they are winning or losing position).

Regarding other ways of solving the tic-tac-toe problem, one could imagine a pseudo-evolutionary multi-agent policy where several agents would play against themselves; the N best would be selected to survive and other will be the result of an offspring base on the best ones where value per node would be the evolutionary parameter: each node would be based on one of the N-best candidate with a probability to be randomly changed. While it seems to converge for the game of Tic-Tac-Toe as it is a simple game, the convergence to an optimal solution for a general game is non-obvious.

2 Multi-armed Bandits

2.1 Untitled

Solution:

$$P(\text{"A greedy action is selected"}) = \frac{\sum_{a \in \mathfrak{A}_{\text{Greedy}}} 1}{|\mathfrak{A}|} = \begin{cases} \epsilon, & \text{if } R(a_1) \neq R(a_2) \\ 1, & \text{else} \end{cases}$$

where $R(a)$ is the reward associated to action $a \in \mathfrak{A}$ and $\mathfrak{A}_{\text{Greedy}}$ is the set of action yielding the maximal reward. In this case, we have that the cardinality of $|\mathfrak{A}|$ is $|\mathfrak{A}| = 2$.

2.2 Bandit example

Solution: A_1 and A_2 are exploratory by definition (first two moves). A_3 and A_4 are potentially greedy as they exploit one of the most rewarding action at this stage (Action 2). A_5 on the other side is clearly an exploratory move as the reward is significantly lower than the average one resulting of action 1 and 2.

Regarding A_3 and A_4 , if an exploratory occurred for at least one fo them, then it would be probably possible that the random action selection at this stage yielded action 2 again. Therefore, the nature of the greediness or exploration cannot be deduced out of the given data.

2.3 Untitled

Solution: Assuming stationary rewards and for a small $\delta > 0$, after a certain time threshold and by the central limit theorem, all agents value functions will only differ by a maximum of δ . While it will be clear to all of them which is the action to exploit, the ones with greater ϵ values will spend a smaller fraction of their time exploiting the optimal reward. Therefore, agent with $\epsilon = 0.01$ will be the one that will collect the most reward on the long run.

2.4 Untitled

Solution: First notice the subtlety in the derivation of the general incremental update rule.

First, note that for general weights $\{\beta\}_{\{i \in I \subset \mathbb{N}\}} \subset \mathbb{R}$ on the rewards, one has

$$Q_{n+1} := \frac{\sum_{i=1}^n \beta R_n}{\sum_{i=1}^n \beta} \quad (2)$$

In particular, if weights $\beta_i = c \in \mathbb{R} \forall i$ are constant, equation (2) becomes

$$Q_{n+1} := \frac{\sum_{i=1}^n \beta_i R_n}{\sum_{i=1}^n \beta_i} \quad (3)$$

$$= \frac{1}{n} \sum_{i=1}^n R_i \quad (4)$$

$$= \frac{1}{n} (R_n + nQ_n - Q_n) \quad (5)$$

$$= Q_n + \frac{1}{n} (R_n - Q_n) \quad (6)$$

$$\iff Q_{n+1} = Q_n + \alpha_n (R_n - Q_n), \text{ where } \alpha_i := \frac{1}{i} \forall n \quad (7)$$

with a learning rate of $\frac{1}{n}$.

However, the question addresses the usage of the update rule stated in equation (7) (derived only for constant weights on the reward) in the context of general non-constant learning rates α_i (which implies that the weight on the rewards are a priori not constant).

Now that we are aware of this theoretical exoticity, we will start investigating the behavior of the update rule prescribed by (7) for non-constant learning rates.

First, one can restate the update rule in a more pleasant form and obtain

$$Q_{n+1} = Q_n + \alpha_n (R_n - Q_n) \quad (8)$$

$$= \alpha_n R_n + (1 - \alpha_n) Q_n \quad (9)$$

An application of the identity (9) $n - 1$ times yields

$$Q_{n+1} = \alpha_n R_n + \sum_{i=1}^{n-1} \alpha_i R_i \cdot \prod_{j=i+1}^n (1 - \alpha_j) + \left(\prod_{i=1}^n (1 - \alpha_i) \right) Q_1 \quad (10)$$

with $Q_1 := \beta_1 R_1$ being the initial weighted reward.

2.5 (Programming)

Solution: Solution code can be found in the following repository: https://github.com/RemyMess/solutions-sutton-rl/tree/master/chapter_2/ex_5.

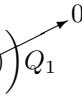
2.6 Mysterious spikes

Solution: For the optimistic methods, the rewards to some actions are set to a non-zero value, in opposition to the realistic ϵ greedy algorithm where all initial reward estimator are 0. At the start, the optimistic agent will select actions it has not previously taken to compute an estimate for the reward. When hitting the optimistically enhanced action with non zero reward, the optimality curve

goes high as the reward is indeed big. Once all the other actions have been selected and an estimator has been computed, the optimistic action is gonna be selected by the agent all the time. In this case, we see that the action that has been enhanced seem to be in reality good one (we cannot say whether it is the best though).

2.7 Unbiased Constant-Set-Size Trick

Solution: (To lighten the notation, I will use $\sigma := \bar{o}$.) First, recall that the incremental update rule derived in (10) for general weights is

$$Q_{n+1} = \beta_n R_n + \sum_{i=1}^{n-1} \beta_i R_i \cdot \prod_{j=i+1}^n (1 - \beta_j) + \left(\prod_{i=1}^n (1 - \beta_i) \right) Q_1 \quad (11)$$


One can immediately deduce the unbiasedness of the iteration rule in our context as $1 - \beta_1 = 0$, implying that the last term of the above sum vanishes. Moreover

Second, one can rewrite the iterative definition for σ as

$$1 - \sigma_n = \gamma - \sigma_{n-1} \cdot \gamma = \gamma(1 - \sigma_{n-1}) \quad (12)$$

with $\sigma_0 = 0$ and $\gamma := 1 - \alpha$.

Clearly, successive application of iteration (12) gives

$$1 - \sigma_n = \gamma^n \quad (13)$$

In addition to that, one can show by a natural induction that $\sigma_k = \sum_{i=0}^{k-1} \gamma^i \alpha$, implying that

$$\beta_n = \left(\sum_{i=0}^{k-1} \gamma^i \right)^{-1} = \frac{1 - \gamma}{1 - \gamma^n} = \frac{\alpha}{1 - \gamma^n} \quad (14)$$

which, when substituted, gives us the full form of the update. The exponential behavior of the coefficient is characteristic of an *exponential recency-weighted average*.

2.8 UCB Spikes

Solution: At the start, all actions are selected according to their preference computed by the UCB. At the start, the uncertainty of all the action is quite high. The agent picks one action after another to reduce the a priori big uncertainty associated to each of them. At step 11, the agent selected an action leading to a high reward. However, as other actions' variance was quite high at this time (not a good estimate on their value) it them, their UCB value was also high and they were therefore selected.(c.f. second term of following equation)

$$UCB(a) := Q_t(a) + c \sqrt{\frac{\ln(t)}{N_t(a)}} \quad (15)$$

for $c \in \mathbb{R}_+$.

After a few other trials, the UCB value of the action at step 11 seem to have become a dominant one relatively to the other ones (i.e. the variance of the other actions got reduced which led the UCB value of the step 11 action to dominate the others).

2.9 Untitled

Solution: Clearly, for $i, j \in \{1, 2\}$ such that $i \neq j$,

$$Pr(\{A_t = A_i\}) = \frac{e^{H_t(a_i)}}{e^{H_t(a_i)} + e^{H_t(a_j)}} \quad (16)$$

$$= \frac{1}{1 + e^{-(H_t(a_i) - H_t(a_j))}} \quad (17)$$

which has the form of a logistic function with growth rate 1 and midpoint $H_t(a_j)$.

2.10 Untitled

Solution: i) Both actions have the same expected reward.

ii) The state spaces will simply consist in the couples $(a, j) \in \{A, B\} \times \{1, 2\}$. To answer about how to achieve the best expectation of success, the answer depends on what time horizon one is considering. For the long run, the best would be to use UCB as the preferences will clearly reach a point where action 1 and 2 respectively for case A and B will be dominant. If we are only interested by the short term reward, an ϵ greedy algorithm with small ϵ value might be the best.

2.11 (Programming)

Will be added soon.

3 Finite Markov Decision Processes

3.1 Untitled

Solution: To be a bit more original, we will consider only Starcraft II agents.

1. "High level control" framework (comparable to instructing a veteran player)

States "Build-order currently in progress", "attacking", "defending", "harassing"

Actions Doing another build order, attacking, defending

Rewards Win

2. **"Low level control" framework (comparable to instructing a machine)**

States Everything (location units, building, unit selected, location camera, on-going upgrades)

Actions Moving units, creating building at specific location, etc..

Rewards Wining gmae / killing units / gathering resources

3. **"Intermediate control" framework (comparable to instructing a beginner)**

States As above minus the position of the buildings and positioning of the units

Actions Creating units and building, gathering ressources, moving units

Rewards Creating units, killing units, gathering resources

3.2 Untitled

Solution: No, it does not apply to non-Markovian decision processes. For example (without entering too deep into the details of the formalisation), consider a robot looking for a mouse somewhere on a $50m^2$ squared area. Depending on the shape of the area, the path the robot is taking matters as the mouse will move away depending on how close the robot might be to it. Moreover, capturing the mouse is only possible if the mouse is scared away towards a wall corner of the area, where the mouse cannot escape.

3.3 Untitled

Solution: There is no "right level": this depends on where one wants to put the control on the system on. It can be required to put set the controls on a high levels if the agent's actions are basic. If the agent is able to understand higher level actions, such as accelerating or simply driving from point A to B, then a higher level formalism for the state space would be preferred. Note that it could be preferred to set the framework on a low level if one is unsure about the mastery of the high level agent's actions or if one would like to allow exploration of new forms of them.

3.4 Untitled

Solution: In the process of being uploaded.