Reinforcement Learning - Assignment 1
Group 3

Davide Belli
11887532
University of Amsterdam
davide.belli@student.uva.nl

Gabriele Cesa
11887524
University of Amsterdam
gabriele.cesa@student.uva.nl

2 Introduction

2.1

The *curse of dimensionality* usually refers to problems which arise when increasing the dimensionality of the space considered. A very common problem is related to the sparsity of the data in high dimensional spaces since the volume increases exponentially with the the number of dimensions. In RL, this is particularly important with a large space of states and actions.

2.2

2.2.1

There are $5 \cdot 5 = 25$ different positions for both the predator and the prey, therefore there are in total $25 \cdot 25 = 625$ different states.

2.2.2

Because to the ciclicity of the toroidal grid, using the exact positions on the grid is worthless. Instead, a better solution is to use the relative position of the prey with respect to the predator.

2.2.3

In this way, the number of states is reduced to only 25.

2.2.4

The advantage is that we don't encode useless information but only the necessary one. Indeed, because of the symmetry of the space, any policy/action of the agent should be independent from the agents' exact positions (translating both agents in any direction of the same distance is well defined and should not change their behavior).

2.2.5

Tic-tac-toe involves a few symmetries as well. For instance, rotating the grid of 90 degrees or flipping it don't change the state of the game. A solution could be to just compute for each grid configuration all its flipped and rotated versions and choose one of them consistently to use as state.

2.3

A non-greedy agent will learn a better policy in the end of the training, since it will learn how to pick the best action for every different state by exploring, eventually, during its learning process, all the possible actions for each state in the game.

2.4

2.4.1

We can add some decay over time on the exploration parameter ϵ (e.g. exponential decay), for example:

$$\begin{aligned} \eta &= 0.95 \\ \epsilon^{(t)} &= \eta^t \epsilon^{(0)} \end{aligned} \tag{1}$$

2.4.2

If the opponent changes strategies this method will not work because, once the exploration parameter ϵ becomes very low, the agent will almost choose every time for the greedy and previously optimal action, and not be able to explore other actions, which now are the optimal ones. To solve this, we could check if the recent reward face an unexpected drop with respect to the previous ones, and, in that case, reset the exploration parameter to a higher value. A more efficient way would be to check for the "convergence" of the expected reward for a state-action pair. This can be done by keeping track of the average and standard deviation of the rewards, and increasing the exploration parameter ϵ once the agent sees a relevant statistical change in the rewards with respect to the expected ones for a certain action-state.

3

3.1

The probability of selecting the greedy action is $(1-\epsilon)$ (the probability of using a greedy policy) plus $\epsilon\frac{1}{n}$ (the probability of taking the random policy but sampling the greedy action), i.e. $(1-\epsilon)+\epsilon\frac{1}{n}$

3.2

Actions A_4 and A_5, since when taking these actions, their expected reward was lower than the best expected reward (0 for action 3 when taking A_4 and 1 for action 2 when taking A_5).

3.3

Consider the set of the two possible actions $\{A, B\}$.

With the optimistic initialization, we start with $Q(A) = Q(B) = 5$. Simulating the agent:

- try A
- get the corresponding reward $+1$ and update $Q(A) = \frac{5+1}{2} = 3$
- try B (notice that A and B could be swapped in these first 2 steps but the result would be the same)
- get the corresponding reward -1 and update $Q(B) = \frac{5-1}{2} = 2$
- $Q(A) > Q(B)$, therefore we greedily try A again
- get the corresponding reward $+1$ and update $Q(A) = \frac{5+1+1}{3} = \frac{7}{3}$

The same values after three steps are obtained if we start picking B.

Conversely, with the pessimistic initialization, we start with $Q(A) = Q(B) = -5$. Simulating the agent, we randomly pick either the action A or the action B as first choice (tie-breaker). Independently of the choice, the respective Q-value will increase (since both rewards are higher than the initial

Q-value. As a consequence, all the successive greedy choices will result in picking that action again and again, and no exploration will be done. In particular, the final Q-values will be:

- $Q(A) = \frac{-5+1+1+1}{4} = -\frac{1}{2}$ and $Q(B) = -5$ if the first action is A
- $Q(A) = -5$ and $Q(B) = \frac{-5-1-1-1}{4} = -2$ if the first action is B

3.4

The highest discounted total reward depends on the initial random choice to break the tie. In our case, with optimistic initialization the total reward is $+1$. With pessimistic initialization we can obtain a total return of -3 (if the initial choice is action B) or of $+3$ (if we first choose A).

3.5

The optimistic initialization leads to a better estimation of the Q-values.

3.6

Optimistic initialization is better for exploration because the less samples we have for an action (i.e. less sure we are about the estimation) the stronger is the bias from the initialization. As a result, actions which haven't been explored much yet will have a higher estimation (since all the rewards will be lower than the initial estimate) and will be more likely to be chosen.

4

4.1

4.1.1

- "A master chess player makes a move. The choice is informed both by planning—anticipating possible replies and counterreplies—and by immediate, intuitive judgments of the desirability of particular positions and moves." **State Space**: Configuration of pieces on the board. **Action Space**: All the feasible movements of the pieces currently on the board. **Reward Signal**: victory or defeat. Possibly, also the "desirability of pieces positions" and the number of captured-lost pieces can be used as partial reward.
- "An adaptive controller adjusts parameters of a petroleum refinery's operation in real time. The controller optimizes the yield/cost/quality trade-off on the basis of specified marginal costs without sticking strictly to the set points originally suggested by engineers." **State Space**: current state of the parameters and value of the marginal costs. **Action Space**: parameters of the operations. **Reward Signal**: the yield/cost/quality trade-off.
- "A gazelle calf struggles to its feet minutes after being born. Half an hour later it is running at 20 miles per hour." **State Space**: the mechanical state of the joints and the muscles, position of the gazelle. **Action Space**: neural signal carrying electric potential in the muscles making them move. **Reward Signal**: satisfaction of the gazelle being able to first stand and then move.
- "A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station. It makes its decision based on the current charge level of its battery and how quickly and easily it has been able to find the recharger in the past." **State Space**: the current charge, position and the robot's memory of the past history. **Action Space**: whether to enter the room or start going back. **Reward Signal**: the amount of garbage the robot is able to collect for each charge with a penality for not getting back to the station before running out of battery.

4.1.2

"An automated traffic light system that controls the traffic lights at a crossing based on the number of cars waiting at the stop. **State Space**: the number of cars for every traffic light, the current color of the lights. **Action Space**: whether to change one or more lights color at the crossing. **Reward signal**: in each time-step, the reward is minus the number of cars waiting in that moment.

4.1.3

"An automated speed check aims to detect vehicles driving faster than the speed limit. To do this, the system takes a picture when it detects some movement in its range." This problem (system) can not be modeled as a Markov Decision Process, since the outcome of every state do not depend only on the current state (position of the car), but also on the previous state (to compute the difference in position and, consequently, the velocity of the car).

4.1.4

We want to model the agent to handle accelerator, handle and steering wheel when we want to solve the problem of interacting with the car where the environment is the observation of the world around us. For example, we want our car to accelerate when the road is straight and free, stop when there is an obstacle, steer when we approach a curve. In this case, the disadvantage is that we only consider how to behave in a low level perspective of the environment (local space), without knowing anything about the route planning.

4.1.5

We want to model the actions are choosing where to drive when we try to tackle a route planning problem. In this case, we want to take optimal actions depending on position, traffic and other information that appear to a high level in our environment (information represented on a graphical map of the world, not (only) based on actual sensor data of the environment around the car). The disadvantage of this is that we are only handling the high-level task of "where to go" bot not the low-level task of "how to go", namely handling the car itself.

4.1.6

To solve the problems mentioned in the two approach, we may want to combine the two agents into a single agents that first updates the route plan given the current state (information from sensors and additional data about the map, traffic). At a second moment, the agent uses the updated planned route combined with sensor data and car state (current acceleration, steering, break) to make actions in order to change the car state (in particular, the three parameters we just mentioned). In this way, both the high-level and low-level tasks of automated driving are handled by our agent.

4.2

4.2.1

For a single episode, using T_i to express the final state for episode i:

$$G_{t,i} = \sum_{k}^{\infty} \gamma^k R_{t+k+1,i} = \sum_{k=t}^{T_i-1} \gamma^k R_{k+1,i} \tag{2}$$

4.2.2

$$\sum_{k=0}^{\infty} \gamma^k = 1 + \gamma + \gamma\gamma + \gamma\gamma^2 + ... + \gamma\gamma^{\infty} < \infty \tag{3}$$

$$\gamma \sum_{k=0}^{\infty} \gamma^k = \gamma + \gamma\gamma + \gamma\gamma^2 + \gamma\gamma^3 + ... + \gamma\gamma^{\infty} < \infty \tag{4}$$

$$\sum_{k=0}^{\infty} \gamma^k - \gamma \sum_{k=0}^{\infty} \gamma^k = 1 \tag{5}$$

$$(1-\gamma) \sum_{k=0}^{\infty} \gamma^k = 1 \tag{6}$$

$$\sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma} \tag{7}$$

4.2.3

The robot is going to get a reward of +1 eventually, so without any discounting the final reward for every episode is +1 independently on the time taken to exit the maze.

4.2.4

If we add a discount with $\gamma < 1$, the positive reward for every episode will change in magnitude depending on the time taken to exit the maze. The agent will learn accordingly which is the optimal way to exit the maze in the fastest way in order to maximize the episodic reward.

4.2.5

A way to change the reward function to solve the learning issue without introducing discounting is to give a small negative reward to every action which is not exiting the maze. The result will be the same to the one of introducing discount, with the agent learning the fastest way out in order to maximize the final episodic reward.

5 Dynamic Programming

5.1

Stochastic:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) \tag{8}$$

Deterministic (greedy):

$$v_\pi^*(s) = q_\pi(s, \pi(s)) \tag{9}$$

5.2

$$\begin{aligned} q_{k+1}(s,a) &= \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')] \qquad (10) \\ &= \sum_{s',r} p(s',r|s,a)[r + \gamma q_k(s', \pi(s')] \qquad (11) \end{aligned}$$

5.3

$$\begin{aligned}\pi_{k+1}(s) &= argmax_a q_\pi(s,a) && (12)\\ &= argmax_a \sum_{s',r} p(s',r|s,a)[r+\gamma v_k(s')] && (13)\\ &= argmax_a \sum_{s',r} p(s',r|s,a)[r+\gamma q_k(s',\pi(s'))] && (14)\end{aligned}$$

5.4

$$q_{k+1}(s,a) = \sum_{s',r} p(s',r|s,a)[r+\gamma \max_{a'} q_k(s',a')] \quad (15)$$

6 Monte Carlo

6.1

6.1.1

first-visit MC $v(s_0) = 5 * \frac{1}{3}(0.9^2 + 0.9^4 + 0.9^3) \approx 3.659$

6.1.2

every-visit MC $v(s_0) = 5 * \frac{1}{3}(\frac{0.9^2+0.9+1}{3} + \frac{0.9^4+0.9^3+0.9^2+0.9+1}{5} + \frac{0.9^3+0.9^2+0.9+1}{4}) \approx 4.304$

6.2

A disadvantage in ordinary importance sampling is that, in case the ratio between the the target and behavior's trajectories are very different, the estimated reward will also be very far from the observed return (lower or higher). As a result, the estimates will have high variance.

6.3

On the other way, the disadvantage of the weighted importance sampling is that the value estimation is biased with expectation $v_b(s)$ instead of $v_\pi(s)$. This means that the episode's trajectory will represent the behavior policy rather than the target policy. This problem is especially noticeable when we use few (or one) trajectory.

7 Temporal Difference Learning (Application)

7.1

Considering for simplicity the update parameter for TD to be $\gamma = 1$ (i.e. no discounting).

TD(0):

t	2	3	4	5	6	7
$V_t(A)$	0.00	-0.30	-0.30	-0.70	-0.89	-0.89
$V_t(B)$	0.00	0.00	0.37	0.37	0.37	0.43

3-step TD:

t	2	3	4	5	6	7
$V_t(A)$	0.00	-0.30	-0.30	-0.87	-0.98	-0.98
$V_t(B)$	0.00	0.00	-0.30	-0.30	-0.30	-0.17

SARSA:

t	2	3	4	5	6	7
$Q_t(A,1)$	0.00	-0.30	-0.30	-0.30	-0.57	-0.57
$Q_t(A,2)$	0.00	0.00	0.00	-0.43	-0.43	-0.43
$Q_t(B,1)$	0.00	0.00	0.40	0.40	0.40	0.40
$Q_t(B,2)$	0.00	0.00	0.00	0.00	0.00	0.10

Q-learning:

t	2	3	4	5	6	7
$Q_t(A,1)$	0.00	-0.30	-0.30	-0.30	-0.53	-0.53
$Q_t(A,2)$	0.00	0.00	0.00	-0.40	-0.40	-0.40
$Q_t(B,1)$	0.00	0.00	0.40	0.40	0.40	0.40
$Q_t(B,2)$	0.00	0.00	0.00	0.00	0.00	0.10

7.2

Given our data and assuming that both the rewards and the transition probabilities are deterministic and constant, we notice that the optimal policy resulting in an infinite reward at infinite time would be the following:

$\pi(A) = 1$
$\pi(B) = 1$

After every 2 time-steps in the episode, the total reward will have increased by 1.

7.3

7.3.1

We assume π_{random} and π_{policy} are used as *behavior policies*. The random policy will result in larger exploration which will lead to explore many state-action pairs with little value since this behavior policy is far from the optimal one. However, this large exploration acts somehow like an exhaustive search; therefore, after enough samples, the method will be able to produce the best values estimations. Conversely, using our deterministic policy will make the algorithm explore only the state-action pairs it defines. As a result, Q-learning will converge to this behavior policy (since other state-action pairs are never seen) and its values estimations will be the one corresponding to this policy (which might not be the optimal one).

7.3.2

The problem with $\pi_{student}$ policy is that it assumes that the rewards are deterministic, without modeling a possible variance in rewards and transition probability. In other words, it corresponds to the greedy policy with (almost) no exploration and does not provide a good estimate of the real state-action values.

The problem with π_{random} is that it ensures a good degree of exploration to estimate the state-action values, but do not exploit the actions resulting in better rewards when used in the real scenario.

7.3.3

Using an ϵ-greedy policy would be beneficial for Q-learning because it would allow for a good trade-off between exploration and exploitation. Indeed, for this particular approach, it is important that the probability distributions of the target and behavior policy are somehow similar to ensure that the samples on which the model is trained are actually matching the ones we will choose in the proposed deterministic target policy.

8 Temporal Difference Learning (Theory)

8.1

By noticing that we can expand and rewrite the update rule as:

$$\begin{aligned}
V_M(S) &= V_{M-1}(S) + \alpha_M[G_M(S) - V_{M-1}(S)] && (16)\\
&= (\alpha - \alpha^M)G_1 + (\alpha - \alpha^M - 1)G_2 + ... + (\alpha - \alpha^2)G_{M-1} + \alpha G_M && (17)\\
&= \alpha G_M + \sum_{k=1}^{M-1} (\alpha - \alpha^{M-k+1})G_k && (18)\\
&\approx \sum_{k=1}^{M} \alpha G_k && (19)
\end{aligned}$$

And then $\alpha_M = \frac{1}{M}$, since for M large enough and $k > 1$, we have $\alpha_M^k = \frac{1}{M^k} \approx 0$

8.2

8.2.1 a

$$\begin{aligned}
\mathbb{E}[\delta_t|S_t = s] &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s') - V^\pi(s)]\\
&= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')] - \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[V^\pi(s)]\\
&= V^\pi(s) - V^\pi(s) \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\\
&= V^\pi(s) - V^\pi(s) \sum_a \pi(a|s)\\
&= V^\pi(s) - V^\pi(s)\\
&= 0
\end{aligned}$$

8.2.2 b

$$\begin{aligned}
\mathbb{E}[\delta_t|S_t = s, A_t = a] &= \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s') - V^\pi(s)]\\
&= \sum_{s',r} p(s',r|s,a)[r + \gamma V^\pi(s')] - \sum_{s',r} p(s',r|s,a) V^\pi(s)\\
&= Q^\pi(s,a) - V^\pi(s) \sum_{s',r} p(s',r|s,a)\\
&= Q^\pi(s,a) - V^\pi(s)
\end{aligned}$$

8.3

8.3.1 a

First, let's expand the expression as before:

$$\begin{aligned}
G_t - V_t(S_t) &= R_{t+1} + \gamma G_{t+1} - V_t(S_t) + \gamma V_{t+1}(S_{t+1}) - \gamma V_{t+1}(S_{t+1})\\
&= R_{t+1} - V_t(S_t) + \gamma V_{t+1}(S_{t+1}) + \gamma(G_{t+1} - V_{t+1}(S_{t+1}))
\end{aligned}$$

Now we have to consider two cases:

- $S_t = S_{t+1} \implies V_{t+1}(S_{t+1}) = V_{t+1}(S_t) = V_t(S_t) + \alpha\delta_t$:

$$\begin{aligned}
G_t - V_t(S_t) &= R_{t+1} - V_t(S_t) + \gamma V_{t+1}(S_{t+1}) + \gamma(G_{t+1} - V_{t+1}(S_{t+1})) \\
&= R_{t+1} - V_t(S_t) + \gamma V_{t+1}(S_t) + \gamma(G_{t+1} - V_{t+1}(S_{t+1})) \\
&= R_{t+1} - V_t(S_t) + \gamma(V_t(S_t) + \alpha\delta_t) + \gamma(G_{t+1} - V_{t+1}(S_{t+1})) \\
&= R_{t+1} - V_t(S_t) + \gamma V_t(S_t) + \gamma\alpha\delta_t + \gamma(G_{t+1} - V_{t+1}(S_{t+1})) \\
&= R_{t+1} - V_t(S_t) + \gamma V_t(S_{t+1}) + \gamma\alpha\delta_t + \gamma(G_{t+1} - V_{t+1}(S_{t+1})) \\
&= \delta_t + \gamma\alpha\delta_t + \gamma(G_{t+1} - V_{t+1}(S_{t+1})) \\
&= (1 + \gamma\alpha)\delta_t + \gamma(G_{t+1} - V_{t+1}(S_{t+1}))
\end{aligned}$$

- $S_t \neq S_{t+1} \implies V_{t+1}(S_{t+1}) = V_t(S_{t+1})$:

$$\begin{aligned}
G_t - V_t(S_t) &= R_{t+1} - V_t(S_t) + \gamma V_{t+1}(S_{t+1}) + \gamma(G_{t+1} - V_{t+1}(S_{t+1})) \\
&= R_{t+1} - V_t(S_t) + \gamma V_t(S_{t+1}) + \gamma(G_{t+1} - V_{t+1}(S_{t+1})) \\
&= \delta_t + \gamma(G_{t+1} - V_{t+1}(S_{t+1}))
\end{aligned}$$

So, we can define:

$$m_t = \begin{cases} 1 & \text{if } S_t \neq S_{t+1} \\ 1 + \alpha\gamma & \text{otherwise} \end{cases}$$

Therefore:

$$\begin{aligned}
G_t - V_t(S_t) &= m_t\delta_t + \gamma(G_{t+1} - V_{t+1}(S_{t+1})) \\
&= m_t\delta_t + \gamma m_{t+1}\delta_{t+1} + \gamma^2(G_{t+2} - V_{t+2}(S_{t+2})) \\
&= m_t\delta_t + \gamma m_{t+1}\delta_{t+1} + \gamma^2 m_{t+2}\delta_{t+2} + \gamma^3(G_{t+3} - V_{t+3}(S_{t+3})) \\
&= m_t\delta_t + \gamma m_{t+1}\delta_{t+1} + \gamma^2 m_{t+2}\delta_{t+2} + ... + \gamma^{T-t-1} m_{T-t-1}\delta_{T-1} + \gamma^{T-t}(G_T - V_T(S_T)) \\
&= m_t\delta_t + \gamma m_{t+1}\delta_{t+1} + \gamma^2 m_{t+2}\delta_{t+2} + ... + \gamma^{T-t-1} m_{T-t-1}\delta_{T-1} + \gamma^{T-t}(0 - 0) \\
&= \sum_{k=0}^{T-t-1} \gamma^k m_{t+k}\delta_{t+k}
\end{aligned}$$

Therefore we need to add $\sum_{k=0}^{T-t-1} \gamma^k (m_{t+k} - 1)\delta_{t+k}$

8.3.2 b

$$
\begin{aligned}
&G_{t:t+n} - V_{t+n-1}(S_t) = \\
&\left(\sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n V_{t+n-1}(S_{t+n})\right) - V_{t+n-1}(S_t) = \\
&\left(\sum_{k=0}^{n-1} \gamma^k R_{t+k+1}\right) - V_{t+n-1}(S_t) + \sum_{k=1}^{n-1} \gamma^k \left(V_{t+n-1}(S_{t+k}) - V_{t+n-1}(S_{t+k})\right) + \gamma^n V_{t+n-1}(S_{t+n}) = \\
&\left(\sum_{k=0}^{n-1} \gamma^k R_{t+k+1}\right) + \left(\sum_{k=0}^{n-1} \gamma^k \left(\gamma V_{t+n-1}(S_{t+k+1}) - V_{t+n-1}(S_{t+k})\right)\right) = \\
&\sum_{k=0}^{n-1} \gamma^k \left(R_{t+k+1} + \gamma V_{t+n-1}(S_{t+k+1}) - V_{t+n-1}(S_{t+k})\right) = \\
&\sum_{k=0}^{n-1} \gamma^k \delta_{t+k}
\end{aligned}
$$

9 Maximization Bias

9.1

	SARSA	Q-learning
$Q(A, left)$	1 *	2
$Q(A, right)$	1.5	1.5
$Q(B, 1)$	1	1
$Q(B, 2)$	1	1
$Q(B, 3)$	2	2
$Q(B, 4)$	0	0

* If we are using a random policy the value is 1. If we use a full greedy policy, the state-action value will be the same as Q-learning (2). If we use and ϵ-greedy policy in between of these corner cases, the value will be between 1 and 2.

9.2

The problem suffers from maximization bias because, when we perform a maximization operation to estimate our Q-value function, we evaluate in an optimistic way the expected future reward of the state A taking action $left$. Indeed, according to the data we observed, the expected reward for action 3 when in state B is 2. As a result, we obtain that $Q(A, left) = 2 > Q(A, right) = 1.5$.

In our case Q-learning suffers from this maximization bias because the $argmax$ operation is performed when estimating the value of the next step S'. SARSA can also suffer from this bias depending on the ϵ parameter chosen for the ϵ-greedy policy. If the policy is very close to greedy, the behavior converges to the one seen in Q-learning, with a maximization bias. If the policy is very far from greedy (then, uniformly random over the set of actions), the state-action values equals to γ and the problem does not suffer from maximization bias

9.3

Double Q-learning can solve this problem alleviating the maximization bias by using two different Q-value functions (Q_1, Q_2) to estimate state-action pairs. For example, let's assume that we split the observation data in 2 subsets (one with the first observation for every action, one with the second observations), where each of them is used to update the estimates of one Q-value function. In particular, we have that Q_1 is trained learned with action-reward datapoints $[0; 2; 2; 0]$, resulting in

$Q_1(B, 1) = 0, \quad Q_1(B, 2) = 2, \quad Q_1(B, 3) = 2, \quad Q_1(B, 4) = 0$

In the same way for Q_2, we obtain:

$Q_2(B, 1) = 2, \quad Q_2(B, 2) = 0, \quad Q_2(B, 3) = 2, \quad Q_2(B, 4) = 0$

Since the Double Q-learning update step takes the expected future reward when updating Q_1 as the Q_2 estimate for the greedy action choice using Q_1 values:

$Q_2(S', argmax_a Q_1(S', a)),$

the expected optimal future reward is either 0 (from action 2, optimal according to Q_1), or 2 (from action 3, also optimal for Q_1), which can be averaged, in convergence, to 1. The same result is obtained for Q_1 updates.

We notice that the Q-values for state A are updated with becomes now:

$Q_1(A, left) = 1, \quad Q_2(A, left) = 1, \quad Q_1(A, right) = 1.5, \quad Q_2(A, right) = 1.5$

As a result, by considering the final state-actiona value functon as the average of the two state-action values functions, we see that the maximization bias is removed, and the optimal action in state A is to go right. The complete picture:

	Q_1	Q_2
$Q(A, l)$	1	1
$Q(A, r)$	1.5	1.5
$Q(B, 1)$	0	2
$Q(B, 2)$	2	0
$Q(B, 3)$	2	2
$Q(B, 4)$	0	0

9.4

Considering that the real distribution of reward is Uniform between either 0 or 2:

	expected values
$Q(A, left)$	1
$Q(A, right)$	1.5
$Q(B, 1)$	1
$Q(B, 2)$	1
$Q(B, 3)$	1
$Q(B, 4)$	1

10 Model-based RL

10.1

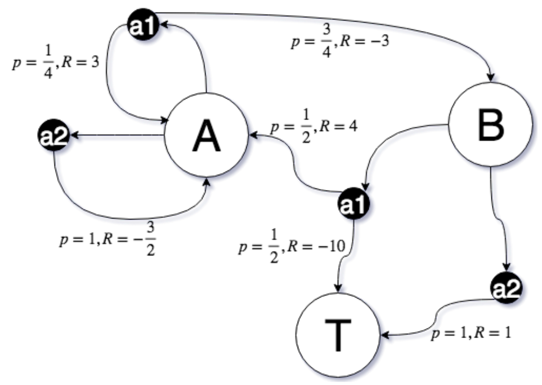

Figure 1: (Maximum Likelihood) Markov Decision Process

10.2

10.2.1 a

The assumption made by Dyna-Q not satisfied by the MPD generating the data is that the environment is deterministic, i.e. given a state and an action, the resulting reward and state are always the same.

10.2.2 b

Dyna-Q could be modified by changing line *(e)* in the algorithm *Tabular Dyna-Q* in section 8.2 of the book such that $Model(S, A)$ implements a distribution over the next state and reward instead of just storing the last pair seen. Consecutively, in the planning steps the pair (R, S') has to be sampled from $Model(S, A)$.

A possible method to build the distribution in $Model(S, A)$ is to build the distribution $p(S'|S, A)$ by counting the number of times a state has followed a pair (S, A), and to compute point estimates of the (conditional) expected rewards $\mathbb{E}[R|S', S, A]$ through sample average. As a result, we can sample $(R, S') \sim Model(S, A)$ by sampling $S' \sim p(S'|S, A)$ and retrieving $R = \mathbb{E}[R|S', S, A]$.

10.2.3 c

The suggested modification can deal with a changing environment if it changes slowly. Changes in the environment will create experience very different from the one learnt in $Model(S, A)$; however, if a lot of experience has been accumulated (many samples have been seen to estimate $p(S'|S, A)$ and $\mathbb{E}[R|S', S, A]$), the model will need a lot of new experience before adapting to the new environment.

A possible solution could be to use an exponential running average for the estimation of the expected rewards. Similarly, we could give to each (S, A, S') sample a weight decreasing with time and use these weights to compute weighted frequencies in order to estimate $p(S'|S, A)$. In this way, we bias these estimations toward the most recent samples and enabling the model to adapt faster to changes in the environment.

10.3

In this question we assume that *Dyna-Q* does not perform its planning process after each step of an episode (as described on the book) but only at the end of an episode (as described in the assignment).

The 2 variants of the 2 algorithms described in the assignment work very similarly. The only difference is at the end of each episode: while the Q-learning variant samples only from the current episode, the Dyna-Q variant samples from all episodes seen so far.

10.3.1

As a result, if the 2 algorithms have seen only one episode (the same) their behaviour is the same. Therefore, we expect the resulting Q-values to be the same after one episode.

10.3.2

However, after the second episode, the Dyna-Q variant will sample also from the first episode whereas the Q-learning variant will only sample from the current one. As a result, their Q-values will differ after this episode.

10.3.3

We expect only the Dyna-Q variant to converge. Indeed, after each episode, the Q-learning variant learns Q-values which are optimal for that episode. As a result, even after many episodes, the Q-values learnt by this algorithm will be the optimal ones for only the last episode, which are unlikely the general optimal ones. Conversely, the Dyna-Q variant learns the Q-values by sampling from all its experience; as a result, assuming a deterministic environment as the original Dyna-Q does, with enough episodes the experience stored will have all the (state, action) pairs (or at least the most likely/relevant). Moreover, the more episodes are seen, the more complete the experience store will be. Therefore, since in the Dyna-Q variant at the end of each episode the Q-values converge to the optimal ones for its whole experience, they will eventually converge to the optimal ones for the MDP.

11 Bonus: Contraction Mapping

11.1

11.1.1

Fixed point is:

$$\begin{aligned} T(x) &= x \\ 1 + \frac{1}{3}x &= x \\ x &= \frac{3}{2} \end{aligned}$$

11.1.2

At fixed point:

$$\begin{aligned} (Tf)(s) &= f(s) \\ -\frac{1}{2}f(s) + g(s) &= f(s) \\ f(s) &= \frac{2}{3}g(s) \end{aligned}$$

11.2

11.2.1

Let $x = \operatorname{argmax}_z f(z)$ and $y = \operatorname{argmax}_z h(z)$. Then, $f(x) \geq f(y)$ and $h(y) \geq h(x)$.

$$
\begin{aligned}
|\max_z f(z) - \max_z h(z)| &= |f(x) - h(y)| \\
&= \max\{f(x) - h(y), h(y) - f(x)\} \\
&\leq \max\{f(x) - h(x), h(y) - f(y)\} \\
&\leq \max\{\max_z(f(z) - h(z)), \max_z(h(z) - f(z))\} \\
&= \max_z |f(z) - h(z)|
\end{aligned}
$$

11.2.2

$$
\begin{aligned}
&||(B^*V_1)(s) - (B^*V_2)(s)||_\infty \\
&= \max_{s\in S} |(B^*V_1)(s) - (B^*V_2)(s)| \\
&= \max_{s\in S} \left|\max_{a\in A}\left(R(s,a) + \gamma \sum_{s'\in S} p(s'|s,a)V_1(s')\right) - \max_{a\in A}\left(R(s,a) + \gamma \sum_{s'\in S} p(s'|s,a)V_2(s')\right)\right| \\
&\leq \max_{s\in S} \max_{a\in A} \left|\left(R(s,a) + \gamma \sum_{s'\in S} p(s'|s,a)V_1(s')\right) - \left(R(s,a) + \gamma \sum_{s'\in S} p(s'|s,a)V_2(s')\right)\right| \\
&= \max_{s\in S} \max_{a\in A} \left|\gamma \sum_{s'\in S} p(s'|s,a)V_1(s') - \gamma \sum_{s'\in S} p(s'|s,a)V_2(s')\right| \\
&= \gamma \max_{s\in S} \max_{a\in A} \left|\sum_{s'\in S} p(s'|s,a)(V_1(s') - V_2(s'))\right| \\
&\leq \gamma \max_{s\in S} \max_{a\in A} \left|\max_{s'\in S}(V_1(s') - V_2(s'))\right| \\
&= \gamma \left|\max_{s'\in S}(V_1(s') - V_2(s'))\right| \\
&\leq \gamma \max_{s'\in S} |V_1(s') - V_2(s')| \\
&= \gamma ||V_1(s) - V_2(s)||_\infty
\end{aligned}
$$