

# Introduction to Artificial Intelligence (INFO8006)

## Exercise session 7

### Markov decision processes

A **Markov decision process** is a tuple  $(\mathcal{S}, \mathcal{A}, P, R)$  such that:

- $\mathcal{S}$  is a set of **states**  $s$ ;
- $\mathcal{A}$  is a set of **actions**  $a$ ;
- $P$  is a (stationary) **transition model** such that  $P(s' | s, a)$  denotes the probability of reaching state  $s'$  if action  $a$  is done in state  $s$ ;
- $R$  is **reward function** that maps immediate (finite) reward values  $R(s)$  obtained in states  $s$ .

### Bellman equation

The **utility** of a state is the immediate reward for that state, plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action:

$$V(s) = R(s) + \gamma \max_a \sum_{s'} P(s' | s, a) V(s').$$

### Value/policy iteration

The **value iteration** algorithm provides a fixed-point iteration procedure for computing the state utilities  $V(s)$ . By denoting  $V_i(s)$  the estimated utility at iteration  $i$  and starting from an initial guess  $V_0(s)$ , we update the estimates to be consistent with **Bellman equation**:

$$V_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s' | s, a) V_i(s')$$

which always converges for  $\gamma < 1$ .

The **policy iteration** algorithm instead directly computes the **policy**  $\pi$  by alternating between **value evaluation** under the current **policy** estimate

$$V_i = V^{\pi_i}$$

and **policy** improvement

$$\pi_{i+1}(s) = \arg \max_a \sum_{s'} P(s' | s, a) V_i(s').$$

### Reinforcement learning

**Reinforcement learning** is used in unknown MDP, i.e. MDP where the **transition model** and the **reward function** are unknown. The goal is still to extract the optimal **policy** by observing or interacting with the environment in order to jointly learn these dynamics and act upon them.

RL can be decomposed in **model-based** and **model-free** methods. The former estimates explicitly the environment and then use this estimation as the empirical MDP whereas the latter do not model the environment explicitly.

## Temporal-difference learning

**Temporal-difference learning** consists in updating  $V^\pi(s)$  each time the agent experiences a transition  $(s, r, a, s')$  under a policy  $\pi$ .

When a transition from  $s$  to  $s'$  occurs, the temporal-difference update steers  $V^\pi(s)$  to better agree with the Bellman equations for a fixed policy, i.e.

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha(r + \gamma V^\pi(s') - V^\pi(s))$$

where  $\alpha$  is the **learning rate**.

## Q-Learning

By defining **Q-values** as  $V(s) = \max_a Q(s, a)$ , we have

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s', a').$$

The state-action-values  $Q(s, a)$  can be learned in a model-free fashion using a temporal-difference method known as **Q-Learning**. Q-Learning consists in updating  $Q(s, a)$  each time the agent experiences a transition. The update equation for TD Q-Learning is

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)).$$

**In session exercises:** Ex. 1, Ex. 4

## Part I

# Making decisions

### Exercise 1 Micro-Blackjack (UC Berkeley CS188, Spring 2014)

In Micro-Blackjack, you repeatedly draw a card (with replacement) that is likely to be a 2, 3 or 4. At each step, if the total score of the cards is lower than 6, you can either “draw” ( $d$ ) or “cash” ( $c$ ). Otherwise, you can only cash. When you cash, the game stops and your utility is equal to your total score (up to 5) plus 1, or zero if you get a total of 6 or higher. Until you cash and after it, you receive no reward. There is no discount ( $\gamma = 1$ ).

1. Formalize Micro-Blackjack as an MDP.

A Markov decision process (MDP) is a tuple  $(S, A, P, R)$ , where  $S$  is a set of reachable states,  $A(s)$  is a set of available actions,  $P(s'|s, a)$  is a (stationary) transition model and  $R(s)$  is a reward function.

In our case,

$$\begin{aligned} S &\subset \mathbb{N} \times \{0, 1\} \\ A(s = (s_1, s_2)) &= \begin{cases} \{d, c\} & \text{if } s_1 < 6 \text{ and } s_2 = 0 \\ \{c\} & \text{if } s_2 = 0 \\ \{\} & \text{otherwise} \end{cases} \\ P(s' = (s'_1, s'_2) | s = (s_1, 0), a) &= \begin{cases} 1 & \text{if } s'_1 = s_1 \text{ and } s'_2 = 1 \text{ and } a = c \\ \frac{1}{3} & \text{if } s'_1 - s_1 \in \{2, 3, 4\} \text{ and } s'_2 = 0 \text{ and } a = d \\ 0 & \text{otherwise} \end{cases} \\ R(s = (s_1, s_2)) &= \begin{cases} s_1 + 1 & \text{if } s_1 < 6 \text{ and } s_2 = 1 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

2. Derive the optimal policy for this MDP.

Let  $(s_0, s_1, \dots, s_l)$  be a sequence of states, *i.e.* a path. Its utility is

$$\sum_{t=0}^l \gamma^t R(s_t),$$

where  $\gamma \in (0, 1]$  is the discount factor. For a policy  $\pi : S \mapsto A$  to be optimal, it should maximize its *expected* utility

$$\begin{aligned} V^\pi(s_0) &= \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \right] \\ &= R(s_0) + \gamma \mathbb{E}[V^\pi(s_1)] \\ &= R(s_0) + \gamma \sum_{s_1} P(s_1 | s_0, \pi(s_0)) V^\pi(s_1), \end{aligned}$$

where the states  $s_{t+1} \sim P(s | s_t, \pi(s_t))$ . Therefore, we define the value  $V(s)$  of a state  $s$  is

its maximum expected utility, *i.e.*

$$\begin{aligned}
V(s) &= V^{\pi^*}(s) \\
&= \max_{\pi} V^{\pi}(s) \\
&= \max_{\pi} R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s') \\
&= R(s) + \gamma \max_{\pi} \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s') \\
&= R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a) \max_{\pi} V^{\pi}(s') \\
&= R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a) V(s'),
\end{aligned}$$

where  $\pi^*$  is the optimal policy

$$\pi^*(s) = \arg \max_{a \in A(s)} \sum_{s'} P(s'|s, a) V(s').$$

To find this optimal policy, we can apply either the *value iteration* or *policy iteration* algorithms.

- In value iteration, we estimate the value  $V(s)$  by value estimates  $V_i(s)$  and update them with the Bellman operator

$$V_{i+1}(s) = (BV_i)(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a) V_i(s')$$

until convergence ( $V_{i+1} = V_i$ ). For our task, it should be noted that all states  $s = (s_1, 1)$  are terminal, meaning that they don't allow to take any actions. Consequently, the value of these states is exactly their reward, *i.e.*

$$V(s = (s_1, 1)) = R(s).$$

Let  $V_0(s) = R(s)$  be our first value estimate.

$s_1$	0	2	3	4	5	6+	0	2	3	4	5	6+
$s_2$			0						1			
$V_0(s)$	0	0	0	0	0	0	1	3	4	5	6	0
$V_1(s)$	1	3	4	5	6	0	1	3	4	5	6	0
$V_2(s)$	$\frac{12}{3}$	$\frac{11}{3}$	4	5	6	0	1	3	4	5	6	0
$V_3(s)$	$\frac{38}{9}$	$\frac{11}{3}$	4	5	6	0	1	3	4	5	6	0
$V_4(s)$	$\frac{38}{9}$	$\frac{11}{3}$	4	5	6	0	1	3	4	5	6	0

Each iteration of this algorithm takes  $\mathcal{O}(|S|^2|A|)$  operations.

- In policy iteration, we estimate the optimal policy  $\pi^*$  by policy estimates  $\pi_i$  and update them with

$$\pi_{i+1}(s) = \arg \max_{a \in A(s)} \sum_{s'} P(s'|s, a) V^{\pi_i}(s'),$$

where  $V^{\pi_i}$  is the unique solution of the simplified Bellman equation

$$V^{\pi_i}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i(s)) V^{\pi_i}(s').$$

However, finding  $V^{\pi_i}$  exactly requires  $\mathcal{O}(|S|^3)$  operations, which is usually intractable. Instead in *modified* policy iteration, we keep value estimates  $V_j$  and update them with the simplified Bellman operator

$$V_{j+1}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi_i) V_j(s').$$

After sufficient convergence ( $V_{j+1} \approx V_j$ ), we update the policy estimate with respect to the last value estimate

$$\pi_j(s) = \arg \max_{a \in A(s)} \sum_{s'} P(s'|s, a) V_j(s')$$

and then start updating the value estimates again. The algorithm stops when the policy estimates have converged. Let  $V_0(s) = R(s)$  be our first value estimate.

$s_1$	0	2	3	4	5	6+	0	2	3	4	5	6+
$s_2$			0					1				
$V_0(s)$	0	0	0	0	0	0	1	3	4	5	6	0
$\pi_0(s)$	$c$	$c$	$c$	$c$	$c$	$c$			-			
$V_1(s)$	1	3	4	5	6	0	1	3	4	5	6	0
$\pi_1(s)$	$d$	$d$	$c$	$c$	$c$	$c$			-			
$V_2(s)$	$\frac{12}{3}$	$\frac{11}{3}$	4	5	6	0	1	3	4	5	6	0
$V_3(s)$	$\frac{38}{9}$	$\frac{11}{3}$	4	5	6	0	1	3	4	5	6	0
$\pi_3(s)$	$d$	$d$	$c$	$c$	$c$	$c$			-			

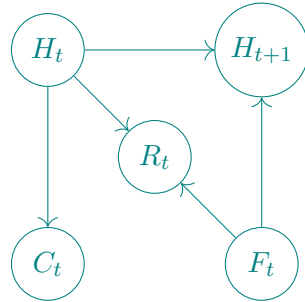
The modified policy iteration algorithm does not necessarily converge faster than the value iteration algorithm, but the simplified Bellman operator is (much) easier and faster to compute than the standard one. One could consider modified policy iteration as a version of value iteration where the optimal actions are “cached” periodically instead of recalculating them at each iteration.

## Exercise 2 Crying Baby Problem

To earn some money you have decided to do babysitting. Tonight you will take care of a 9 months old baby. His parents asked you to feed him when he is hungry. Because you are a super-babysitter you know from your experience that the probability of the baby crying when he is hungry is equal to 0.8 and to cry when he is not hungry is equal to 0.1. The cost of feeding the baby is equal to 5 \$ whereas if you don't feed him while he is hungry it costs you 10 \$ (because the parent will be upset and will reduce your pay). You will have many things to do tonight and so you decide to optimize your time by just checking every hour whether the baby is crying. Between two intervals of time you assume that the baby has a probability of 0.2 to get hungry. You can also assume that the baby has a uniform probability of being hungry when you start your babysitting.

1. Draw the dynamic Bayesian network as well as the conditional probability tables associated with your babysitting of tonight.

Let  $H_t \in \{0,1\}$  indicate whether the baby is hungry at time  $t$ ,  $C_t \in \{0,1\}$  the baby is crying at time  $t$  and  $F_t \in \{0,1\}$  the fact that you fed the baby at time  $t$ . Your cost is  $R_t \in \{0, -5, -10\}$ .



$H_t$	$F_t$	$P(C_t = 1 H_t)$	$P(H_{t+1} = 1 H_t, F_t)$	$R_t(H_t, F_t)$
0	0	0.1	0.2	0
1	0	0.8	1	-10
0	1	-	0	-5
1	1	-	0	-5

For completeness we also have to define the prior  $P(H_0 = 1) = 0.5$ . Note that  $R_t$  is a deterministic function of  $H_t$  and  $F_t$  instead of a distribution.

2. Give the sequence of belief states if you assume the sequence of observation-action pairs  $(0, 0), (1, 1), (0, 0), (0, 0)$ .

In this problem, the environment is only *partially* observable (POMDP). You, the agent, does not know the current state  $s = H_t$ . Instead, you collect an observation  $e = C_t$  through a sensor model  $P(e|s)$  which allows to reason about the unknown state  $s$  and to take an informed action  $a = F_t$ .

If  $b$  is the current belief state (a distribution over the state space  $S$ ) and the agent does an action  $a$  and observes  $e'$ , then

$$\begin{aligned}
 b'(s') &= \sum_s P(s'|s, a, e')b(s) \\
 &= \alpha P(e'|s') \sum_s P(s'|s, a)b(s)
 \end{aligned}$$

is the next belief state. Within the formulation of our problem, this gives

$$b_{t+1}(h_{t+1}) = \alpha P(c_{t+1}|h_{t+1}) \sum_{h_t} P(h_{t+1}|h_t, f_t) b_t(h_t),$$

with the initial belief

$$\begin{aligned} b_0 &= P(H_0|c_0) \\ &= \alpha P(c_0|H_0)P(H_0) \end{aligned}$$

Then, knowing from the statement that  $c_{0:3} = f_{0:3} = (0, 1, 0, 0)$ , we have

$$\begin{aligned} b_0 &= \alpha_0 \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix} \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} \approx \begin{pmatrix} 0.8181 \\ 0.1819 \end{pmatrix} \\ b_1 &= \alpha_1 \begin{pmatrix} 0.1 & 0 \\ 0 & 0.8 \end{pmatrix} \begin{pmatrix} 0.8 & 0 \\ 0.2 & 1 \end{pmatrix} b_0 \approx \begin{pmatrix} 0.1915 \\ 0.8085 \end{pmatrix} \\ b_2 &= \alpha_2 \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} b_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ b_3 &= \alpha_3 \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix} \begin{pmatrix} 0.8 & 0 \\ 0.2 & 1 \end{pmatrix} b_2 \approx \begin{pmatrix} 0.9474 \\ 0.0526 \end{pmatrix}. \end{aligned}$$

3. From the sequence of belief states, determine the sequence of expected rewards.

The expected reward of a belief state  $b$  is defined as

$$\rho(b) = \sum_s b(s)R(s).$$

In our case, the expected reward also depend on the action, that is

$$\rho_t = \sum_{h_t} R_t(h_t, f_t) b(h_t) = R_t(H_t, f_t)^T b_t.$$

Then, by substitution,

$$\begin{aligned} \rho_0 &= \begin{pmatrix} 0 & -10 \end{pmatrix} b_0 \approx -1.819 \\ \rho_1 &= \begin{pmatrix} -5 & -5 \end{pmatrix} b_1 = -5 \\ \rho_2 &= \begin{pmatrix} 0 & -10 \end{pmatrix} b_2 = 0 \\ \rho_3 &= \begin{pmatrix} 0 & -10 \end{pmatrix} b_3 \approx -0.526. \end{aligned}$$

### Exercise 3 Pursuit Evasion (UC Berkeley CS188, Spring 2014)

Pacman is trapped in the following 2 by 2 maze with a hungry ghost. When it is his turn to move, Pacman must move one step horizontally or vertically to a neighboring square. When it is the ghost's turn, he must also move one step horizontally or vertically. The ghost and Pacman alternate moves. After every move (by either the ghost or Pacman), if Pacman and the ghost occupy the same square, Pacman is eaten and receive utility  $-100$ . Otherwise, he receives utility of 1. The ghost attempts to minimize the utility that Pacman receives. Pacman makes the first move. The game is not guaranteed to terminate.



For example, with a discount factor of  $\gamma = 1$ , if Pacman moves right, ghost moves down, then Pacman moves left, Pacman earns a reward of 1 initially ( $R(s_0)$ ) and after the two first moves and  $-100$  after the last move, for a total utility of  $-97$ .

1. Assume a discount factor  $\gamma = 0.5$ , where the discount factor is applied once every time either Pacman or the ghost moves. What is the Minimax value of the complete (infinite) game?

If we assume the game starts from the displayed state  $s_0$ , Pacman has always an action to get out of reach of the ghost. Hence, if Pacman is optimal, which is true by definition in Minimax, he will never get eaten. Therefore, we can compute the initial state value

$$V(s_0) = \sum_{t=0}^{\infty} \gamma^t \underbrace{R(s_t)}_1 = \lim_{t \rightarrow \infty} \frac{1 - \gamma^{t+1}}{1 - \gamma} = \frac{1}{1 - \gamma} = 2,$$

without actually building the (infinite) Minimax tree.

2. Why are the value/policy iteration algorithms superior to Minimax for solving this game?  
If  $\gamma < 1$ , we know that the value/policy iteration algorithms converge toward a fixed point, even if the game does not terminate. Conversely, Minimax is not guaranteed to terminate due to its recursivity.

This game is similar to an MDP because rewards are earned at every timestep. However, it is also an adversarial game involving decisions by two agents. Let  $s$  be the state (*e.g.* the positions of Pacman and the ghost and who is playing) and let  $A(s)$  be the set of actions available to the player in state  $s$ . Let  $s' = T(s, a)$  denote the successor of a state  $s$  resulting from the action  $a$ . Finally, let  $R(s)$  denote the utility received after moving to state  $s$ .

4. Write down an expression (analogous to Bellman equation) for  $V(s)$ , the value of the current state if it is Pacman's turn.

$$V(s) = R(s) + \gamma \max_{a \in A(s)} \left[ R(T(s, a)) + \gamma \min_{a' \in A(T(s, a))} V(T(T(s, a), a')) \right]$$



## Part II

# Reinforcement learning

### Exercise 4 Q-learning

An agent is in an unknown environment where there are three states  $\{A, B, C\}$  and two actions  $\{0, 1\}$ . We are given the following tuples  $(s, a, r, s')$ , generated by taking actions in the environment.

$s$	$a$	$r$	$s'$
$A$	0	+2	$A$
$C$	1	-2	$A$
$B$	1	+1	$B$
$A$	0	-1	$B$
$B$	1	-2	$C$
$C$	0	+4	$B$
$B$	0	+1	$A$

Assuming a discount factor  $\gamma = 0.5$  and a learning rate  $\alpha = 0.75$ ,

1. Apply the  $Q$ -learning algorithm to obtain state-action-value  $Q(s, a)$  estimates. Estimates are initialized to 0.

As seen in the previous exercise, given trial trajectories of some policy  $\pi$ , it is possible to estimate its expected utility  $V^\pi$  even without knowing the transition model  $P(s'|s, a)$  or the reward function  $R(s)$  of the environment. However, estimating  $V^\pi$  only evaluates the quality of  $\pi$ , but does not describe how to improve it.

By definition, the optimal policy  $\pi^*$  is the one that maximizes the expect utility of the state  $s$ , *i.e.*

$$\pi^*(s) = \arg \max_a \sum_{s'} P(s'|s, a) V(s')$$

where  $V(s) = \max_\pi V^\pi(s)$  is the state-value of  $s$ . Unfortunately, even if we knew  $V$ , we could not find the optimal actions without knowing the transition model. However, if we knew the state-action-value  $Q(s, a)$  of taking action  $a$  in state  $s$ , we would be able to select the optimal action

$$\pi^*(s) = \arg \max_a Q(s, a),$$

where

$$\begin{aligned} Q(s, a) &= R(s) + \gamma \sum_{s'} P(s'|s, a) V(s') \\ &= R(s) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a') \end{aligned}$$

since  $V(s) = \max_a Q(s, a)$ . Fortunately, state-action-values  $Q(s, a)$  can be learned in a model-free fashion using the  $Q$ -learning algorithm. Similarly to temporal-difference learning,

in  $Q$ -learning, we perform stochastic gradient descent updates

$$\begin{aligned}\hat{Q}(s, a) &\leftarrow \hat{Q}(s, a) - \alpha \left( \hat{Q}(s, a) - r - \gamma \max_{a'} \hat{Q}(s', a') \right) \\ &\leftarrow (1 - \alpha) \hat{Q}(s, a) + \alpha \left( r + \gamma \max_{a'} \hat{Q}(s', a') \right)\end{aligned}$$

from observed tuples  $(s, r, a, s')$ .

In our case, the sequence of updates would be

$$\begin{aligned}\hat{Q}(A, 0) &\leftarrow (1 - \alpha) \hat{Q}(A, 0) + \alpha \left( 2 + \gamma \max\{\hat{Q}(A, 0), \hat{Q}(A, 1)\} \right) = \frac{1}{4} 0 + \frac{3}{4} \left( 2 + \frac{1}{2} 0 \right) = \frac{+3}{2} \\ \hat{Q}(C, 1) &\leftarrow (1 - \alpha) \hat{Q}(C, 1) + \alpha \left( -2 + \gamma \max\{\hat{Q}(A, 0), \hat{Q}(A, 1)\} \right) = \frac{1}{4} 0 + \frac{3}{4} \left( -2 + \frac{1}{2} \frac{3}{2} \right) = \frac{-15}{16} \\ \hat{Q}(B, 1) &\leftarrow (1 - \alpha) \hat{Q}(B, 1) + \alpha \left( 2 + \gamma \max\{\hat{Q}(B, 0), \hat{Q}(B, 1)\} \right) = \frac{1}{4} 0 + \frac{3}{4} \left( 1 + \frac{1}{2} 0 \right) = \frac{+3}{4} \\ \hat{Q}(A, 0) &\leftarrow (1 - \alpha) \hat{Q}(A, 0) + \alpha \left( -1 + \gamma \max\{\hat{Q}(B, 0), \hat{Q}(B, 1)\} \right) = \frac{1}{4} \frac{3}{2} + \frac{3}{4} \left( -1 + \frac{1}{2} \frac{3}{4} \right) = \frac{-3}{32} \\ \hat{Q}(B, 1) &\leftarrow \dots\end{aligned}$$

It should be noted that, as for TD learning, we are not limited to perform only one update for each tuple. Reusing tuples several times (in other orders) is a good way to reach convergence faster while generating less trials.

2. We now switch to a feature-based estimator  $\hat{Q}(s, a) = w_0 + w_1 f_1(s, a)$ , with  $f_1(s, a) = 2a - 1$ . Starting from weights  $w_0 = w_1 = 0$ , update the weights according to the approximate  $Q$ -learning algorithm.

We would like our estimates  $\hat{Q}(s, a)$  to satisfy the Bellman equation, *i.e.* to minimize

$$L = \mathbb{E}_{s'|s,a} \left[ \left( R(s) + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a) \right)^2 \right].$$

To reach this objective we follow the opposite of its gradient

$$\nabla_w L = 2 \mathbb{E}_{s'|s,a} \left[ \left( \hat{Q}(s, a) - R(s) - \gamma \max_{a'} \hat{Q}(s', a') \right) \nabla_w \hat{Q}(s, a) \right]$$

with respect to the weights<sup>1</sup>  $w$  of  $\hat{Q}(s, a)$ . Once again, due to the expectation, we are forced to use a stochastic approximation of the gradient from observed tuples  $(s, a, r, s')$  and to update the weights as

$$w \leftarrow w - \alpha \left( \hat{Q}(s, a) - r - \gamma \max_{a'} \hat{Q}(s', a') \right) \nabla_w \hat{Q}(s, a),$$


where  $\alpha$  is the learning rate. In our case,

$$\nabla_w \hat{Q}(s, a) = \begin{pmatrix} \partial_{w_0} \\ \partial_{w_1} \end{pmatrix} (w_0 + w_1 f_1(s, a)) = \begin{pmatrix} 1 \\ f_1(s, a) \end{pmatrix} = \begin{pmatrix} 1 \\ 2a - 1 \end{pmatrix}$$

and the two first weight updates would be

$$\begin{aligned}w &\leftarrow w - \alpha \left( \hat{Q}(A, 0) - 2 - \gamma \max\{\hat{Q}(A, 0), \hat{Q}(A, 1)\} \right) \nabla_w \hat{Q}(A, 0) \\ &\leftarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix} - \frac{3}{4} \left( -2 - \frac{1}{2} \max\{0, 0\} \right) \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} \frac{+3}{2} \\ \frac{-3}{2} \end{pmatrix} \\ w &\leftarrow w - \alpha \left( \hat{Q}(C, 1) + 2 - \gamma \max\{\hat{Q}(A, 0), \hat{Q}(A, 1)\} \right) \nabla_w \hat{Q}(C, 1) \\ &\leftarrow \begin{pmatrix} \frac{+3}{2} \\ \frac{-3}{2} \end{pmatrix} - \frac{3}{4} \left( 2 - \frac{1}{2} \max\{0, 3\} \right) \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{+9}{8} \\ \frac{-15}{8} \end{pmatrix}.\end{aligned}$$

## Exercise 5 Passive RL

3	-6	-1	+5
2			
1		-8	+3
	1	2	3

Consider the grid-world given above and an agent who is trying to learn the optimal policy. The agent starts from the bottom-left corner and can take the actions **north** ( $N$ ), **south** ( $S$ ), **west** ( $W$ ) and **east** ( $E$ ). Rewards are only awarded for reaching the terminal (shaded) states. You observe the following trials, whose trajectories are sequences of tuples  $(s_t^i, r_t^i, a_t^i, s_{t+1}^i)$ .

$t$	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
0	(1, 1), 0, $N$ , (1, 2)	(1, 1), 0, $N$ , (1, 2)	(1, 1), 0, $N$ , (1, 2)	(1, 1), 0, $N$ , (1, 2)	(1, 1), 0, $N$ , (1, 2)
1	(1, 2), 0, $E$ , (2, 2)	(1, 2), 0, $E$ , (2, 2)	(1, 2), 0, $E$ , (2, 2)	(1, 2), 0, $E$ , (2, 2)	(1, 2), 0, $E$ , (2, 2)
2	(2, 2), 0, $N$ , (2, 3)	(2, 2), 0, $E$ , (3, 2)	(2, 2), 0, $S$ , (2, 1)	(2, 2), 0, $E$ , (3, 2)	(2, 2), 0, $E$ , (3, 2)
3	(2, 3), -1, $\emptyset$ , $\emptyset$	(3, 2), 0, $N$ , (3, 3)	(2, 1), -8, $\emptyset$ , $\emptyset$	(3, 2), 0, $W$ , (2, 2)	(3, 2), 0, $S$ , (3, 1)
4		(3, 3), +5, $\emptyset$ , $\emptyset$		(2, 2), 0, $N$ , (2, 3)	(3, 1), +3, $\emptyset$ , $\emptyset$
5				(2, 3), -1, $\emptyset$ , $\emptyset$	

Assuming a discount factor  $\gamma = 1$ ,

1. Perform direct utility estimation of the expected utilities  $V^\pi(s)$ , given the four first trials.

In this setting, the transition model  $P(s'|s, a)$  and the reward function  $R(s)$  are unknown. We wish to learn the expected utility  $V^\pi$  of the policy  $\pi$  without modeling the environment, *i.e.* without building approximates  $\hat{P}(s'|s, a)$  and  $\hat{R}(s)$ . The principle of direct utility estimation is to approximate  $V^\pi(s)$  by the average utility  $\hat{V}(s)$  of the state  $s$  within all trial trajectories, *i.e.*

$$\hat{V}(s) = \frac{1}{N(s)} \sum_{(i,j) \in I(s)} \sum_{t=0}^{\infty} \gamma^t r_{j+t}^i \approx V^\pi(s) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t) \right] \Big|_{s_0=s}$$

where  $I(s) = \{(i, j) : s_j^i = s\}$  indexes the occurrences of  $s$  and  $N(s) = |I(s)|$  is the number of occurrences of  $s$ . For examples, in our case and excluding the fifth trial,

$$\begin{aligned} I((1, 1)) &= \{(1, 0), (2, 0), (3, 0), (4, 0)\} \\ \hat{V}((1, 1)) &\approx \frac{1}{4} \left( -1\gamma^3 + 5\gamma^4 - 8\gamma^3 - 1\gamma^5 \right) = \frac{-5}{4} \\ I((2, 2)) &= \{(1, 2), (2, 2), (3, 2), (4, 2), (4, 4)\} \\ \hat{V}((2, 2)) &\approx \frac{1}{5} \left( -1\gamma^1 + 5\gamma^2 - 8\gamma^1 - 1\gamma^3 - 1\gamma^1 \right) = \frac{-6}{5} \\ I((2, 3)) &= \{(1, 3), (4, 5)\} \\ \hat{V}((2, 3)) &\approx \frac{1}{2} \left( -1\gamma^0 - 1\gamma^0 \right) = -1. \end{aligned}$$

<sup>1</sup>It should be noted that the target  $R(s) + \gamma \max_{a'} \hat{Q}(s', a')$  should not be affected by the gradient operation as it is considered constant in the objective. In practice, this is not true as modifying the weights in a parametric function  $\hat{Q}$  does so for all state-action pairs  $(s, a)$  at once. This generally makes the approximate  $Q$ -learning algorithm very unstable.

Importantly, since the agent hasn't reached the states (1, 3) and (3, 3) yet, it is not possible to estimate their expected utility. Instead, we assume a default value of 0.

3	0	-1	+5
2	$-\frac{5}{4}$	$-\frac{6}{5}$	+2
1	$-\frac{5}{4}$	-8	0
	1	2	3

- Update the estimated expected utilities with respect to the fifth trial using temporal-difference learning. Assume a learning rate  $\alpha = 0.5$ .

Direct utility estimation misses the fact that the expected utilities for different states are not independent, since they obey the Bellman equation for a fixed policy

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^\pi(s').$$

Then, we would like our estimates  $\hat{V}(s)$  to verify the Bellman equation, *i.e.* to minimize

$$L = \mathbb{E}_{s'|s} \left[ \left( R(s) + \gamma \hat{V}(s') - \hat{V}(s) \right)^2 \right].$$

One way to reach this objective is to follow the opposite of its gradient

$$\nabla L = 2 \mathbb{E}_{s'|s} \left[ \hat{V}(s) - R(s) - \gamma \hat{V}(s') \right]$$

with respect to  $\hat{V}(s)$ , *i.e.* to perform *gradient descent*. Unfortunately, due to the expectation, we don't have access to the true gradient. Instead, we use a *stochastic approximation* of the gradient from observed<sup>2</sup> tuples  $(s, r, a, s')$  and update  $\hat{V}(s)$  as

$$\begin{aligned} \hat{V}(s) &\leftarrow \hat{V}(s) - \alpha \left( \hat{V}(s) - r - \gamma \hat{V}(s') \right) \\ &\leftarrow (1 - \alpha) \hat{V}(s) + \alpha \left( r + \gamma \hat{V}(s') \right) \end{aligned}$$

where  $\alpha$  is the learning rate. For a fixed learning rate, the average of  $\hat{V}(s)$  converge towards (oscillates around) the true expected utility  $V^\pi(s)$ . For a (slowly) decreasing learning rate,  $\hat{V}(s)$  itself converges to  $V^\pi(s)$ . For historical reasons, this algorithm is called temporal-difference (TD) learning.

Following the trajectory of the fifth trial, we perform the updates

$$\begin{aligned} V^\pi((1, 1)) &\leftarrow (1 - \alpha)V^\pi((1, 1)) + \alpha(0 + \gamma V^\pi((1, 2))) = \frac{1}{2} \frac{-5}{4} + \frac{1}{2} \frac{-5}{4} = \frac{-5}{4} \\ V^\pi((1, 2)) &\leftarrow (1 - \alpha)V^\pi((1, 2)) + \alpha(0 + \gamma V^\pi((2, 2))) = \frac{1}{2} \frac{-5}{4} + \frac{1}{2} \frac{-6}{5} = \frac{-49}{40} \\ V^\pi((2, 2)) &\leftarrow (1 - \alpha)V^\pi((2, 2)) + \alpha(0 + \gamma V^\pi((3, 2))) = \frac{1}{2} \frac{-6}{5} + \frac{1}{2} 2 = \frac{+4}{10} \\ V^\pi((3, 2)) &\leftarrow (1 - \alpha)V^\pi((3, 2)) + \alpha(0 + \gamma V^\pi((3, 1))) = \frac{1}{2} 2 + \frac{1}{2} 0 = +1 \\ V^\pi((3, 1)) &\leftarrow (1 - \alpha)V^\pi((3, 1)) + \alpha(3 + 0) = \frac{1}{2} 0 + \frac{1}{2} 3 = \frac{+3}{2}. \end{aligned}$$

<sup>2</sup>We are not limited to perform only one update for each tuple  $(s, r, a, s')$ . Indeed, as we assume the policy to be stationary, past trials are as likely to happen again as more recent ones. Therefore, reusing tuples several times (in other orders) is a good way to improve estimates while generating less trials.

3	0	-1	+5
2	$-\frac{49}{40}$	$+\frac{4}{10}$	+1
1	$-\frac{5}{4}$	-8	$+\frac{3}{2}$
	1	2	3

## Exercise 6 Football

ULiège's football team is playing against UCL's team next week. With a lot of losses this season, Liège needs to improve their attack strategy to win the game and increase their popularity. Luckily, the team captain follows INFO8006 and knows how to model the attack as a Markov Decision Process. The captain considers four states **close** (C), **away** (A), **fail** (F), and **goal** (G), and two actions **pass** (P) and **shoot** (S). Although the transition probabilities are unsure, the possible transitions  $(s, a, s')$  are known. To each transition is associated an increase/decrease of the team's popularity.

$s$	$a$	$s'$	$R(s, a, s')$
C	P	C	+1
C	P	A	-1
C	P	F	-2
C	S	C	+3
C	S	F	-5
C	S	G	+10
A	P	C	+2
A	P	A	0
A	P	F	-3
A	S	C	+3
A	S	F	-10
A	S	G	+20

The current strategy of the team is to always shoot. Last match, they had several attack opportunities, resulting in the following actions.

$s$	C	C	C	C	A	A	A	A
$a$	S	S	S	S	S	S	S	S
$s'$	G	C	G	F	F	C	F	F

Assuming a discount factor  $\gamma = 0.75$  and a learning rate  $\alpha = 0.25$ ,

1. Build an estimator of the transition model  $P(s'|s, a)$  and, from it, determine the expected utility  $V^\pi$  of the team's current policy  $\pi$ . Given the previous table, we can estimate the transition probabilities for  $s \in \{\text{C, A}\}$  and  $a = \text{S}$ .

We can now determine the expected utility  $V^\pi$  of the team's current policy  $\pi(s) = \text{S}$  using the following Bellman operator

$$V_{i+1}^\pi(s) = \sum_{s'} P(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma V_i^\pi(s')],$$

$s$	$a$	$P(s' s, a)$			
		C	A	F	G
C	S	0.25	0	0.25	0.5
A	S	0.25	0	0.75	0

with  $V_0^\pi(s) = 0$  for all state  $s$ . Because the states **F** and **G** are terminal, their expected utility remains constant and we only have to update **C** and **A**. For the first iteration,

$$\begin{aligned} V_1^\pi(\mathbf{C}) &= \sum_{s'} P(s'|\mathbf{C}, \mathbf{S}) [R(\mathbf{C}, \mathbf{S}, s') + \gamma V_0^\pi(s')] \\ &= 0.25(3 + \gamma 0) + 0.25(-5 + \gamma 0) + 0.5(10 + \gamma 0) = 4.5 \end{aligned}$$

and

$$\begin{aligned} V_1^\pi(\mathbf{A}) &= \sum_{s'} P(s'|\mathbf{A}, \mathbf{S}) [R(\mathbf{A}, \mathbf{S}, s') + \gamma V_0^\pi(s')] \\ &= 0.25(3 + \gamma 0) + 0.75(-10 + \gamma 0) = -6.75. \end{aligned}$$

We repeat the same procedure until  $V_{i+1}^\pi(s) \approx V_i^\pi(s)$  for all state  $s$ .

$s$	C	A	F	G
$V_0^\pi(s)$	0	0	0	0
$V_1^\pi(s)$	4.5	-6.75	0	0
$V_2^\pi(s)$	5.3437	-5.9063	0	0
$V_3^\pi(s)$	5.5019	-5.7480	0	0
$V_4^\pi(s)$	5.5316	-5.7183	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$V_8^\pi(s)$	5.5385	-5.7115	0	0

The captain found the tapes of the previous season where they had much more success. Together with the team, the captain selects the following instructive actions.

$s$	C	C	A	A	C	A	A	C
$a$	S	S	S	P	P	P	S	P
$s'$	G	C	F	C	C	C	F	F

- Given the selected tuples, apply the  $Q$ -learning algorithm to obtain state-action-value  $Q(s, a)$  estimates. Estimates are initialized to 0.

In  $Q$ -learning, the state-action-value  $Q(s, a)$  estimates are updated following

$$\begin{aligned} Q(s, a) &\leftarrow Q(s, a) - \alpha \left( Q(s, a) - r - \gamma \max_{a'} Q(s', a') \right) \\ &\leftarrow (1 - \alpha) Q(s, a) + \alpha \left( r + \gamma \max_{a'} Q(s', a') \right) \end{aligned}$$

for observed tuples  $(s, r, a, s')$ . In our case, we have a list a 16  $(8 + 8)$  tuples  $(s, a, s')$  and we have access to the reward function  $R(s, a, s')$ . Using the tuples in arbitrary order, we

have

$$\begin{aligned}Q(\mathbf{C}, \mathbf{S}) &\leftarrow (1 - \alpha)Q(\mathbf{C}, \mathbf{S}) + \alpha \left( R(\mathbf{C}, \mathbf{S}, \mathbf{G}) + \gamma \max_{a'} Q(\mathbf{G}, a') \right) \\ &\leftarrow 0.75 \times 0 + 0.25(10 + 0) = 2.5\end{aligned}$$

$$\begin{aligned}Q(\mathbf{C}, \mathbf{S}) &\leftarrow (1 - \alpha)Q(\mathbf{C}, \mathbf{S}) + \alpha \left( R(\mathbf{C}, \mathbf{S}, \mathbf{C}) + \gamma \max_{a'} Q(\mathbf{C}, a') \right) \\ &\leftarrow 0.75 \times 2.5 + 0.25(3 + 0.75 \max\{2.5, 0\}) = 3.094\end{aligned}$$

$$\begin{aligned}Q(\mathbf{A}, \mathbf{S}) &\leftarrow (1 - \alpha)Q(\mathbf{A}, \mathbf{S}) + \alpha \left( R(\mathbf{A}, \mathbf{S}, \mathbf{F}) + \gamma \max_{a'} Q(\mathbf{F}, a') \right) \\ &\leftarrow 0.75 \times 0 + 0.25(-10 + 0) = -2.5\end{aligned}$$

$$\begin{aligned}Q(\mathbf{A}, \mathbf{P}) &\leftarrow (1 - \alpha)Q(\mathbf{A}, \mathbf{P}) + \alpha \left( R(\mathbf{A}, \mathbf{P}, \mathbf{C}) + \gamma \max_{a'} Q(\mathbf{C}, a') \right) \\ &\leftarrow 0.75 \times 0 + 0.25(2 + 0.75 \max\{3.094, 0\}) = 4.320\end{aligned}$$

$$Q(\mathbf{C}, \mathbf{P}) \leftarrow \dots$$

3. Determine the optimal policy according to the state-action-value estimates.

$$\pi(s) = \begin{cases} \mathbf{S} & \text{if } s = \mathbf{C} \\ \mathbf{P} & \text{if } s = \mathbf{A} \end{cases}$$

# Supplementary exercises

## Exercise 7 (AIMA, Ex 16.15)

Consider a student, Sam, who has the choice to buy or not buy a textbook for a course. We'll model this as a decision problem with one Boolean decision node,  $B$ , indicating whether the agent chooses to buy the book, and two Boolean chance nodes,  $M$ , indicating whether the student has mastered the material of the course, and  $F$ , indicating whether the student fails the course. There is also a utility node,  $U$ . Sam has an additive utility function: 0 for not buying the book and  $-100$  \$ for buying it; and 0 for failing the course and 2000 \$ for passing it. Sam's conditional probability estimates are as follows:

$B$	$M$	$P(M B)$	$P(F = 1 B, M)$
0	0	0.3	0.7
1	0	0.1	0.5
0	1	0.7	0.2
1	1	0.9	0.1

1. Draw the decision network for this problem.
2. Compute the expected utility of buying the book and of not buying it.
3. What should Sam do?



## Exercise 8 (AIMA, Ex 17.6)

The slides of the course states that the Bellman operator

$$(BV)(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a) V(s')$$

is a contraction.

1. Show that, for any two functions  $f : \mathbb{R} \mapsto \mathbb{R}$  and  $g : \mathbb{R} \mapsto \mathbb{R}$ ,

$$|\max_x f(x) - \max_x g(x)| \leq \max_x |f(x) - g(x)|.$$

2. Apply the previous result to prove that the Bellman operator is a contraction of factor  $\gamma$ , that is

$$\|(BV) - (BV')\|_\infty \leq \gamma \|V - V'\|_\infty.$$

---

The infinity norm of a function  $f : \mathbb{R} \mapsto \mathbb{R}$  is defined as

$$\|f\|_\infty = \sup_x |f(x)|.$$

## Exercise 9 (AIMA, Ex 17.8)

Consider the  $3 \times 3$  world shown in Figure ???. The transition model is the same as in the  $4 \times 3$  ????: 80% of the time the agent goes in the direction it selects; the rest of the time it moves at right angles to the intended direction.

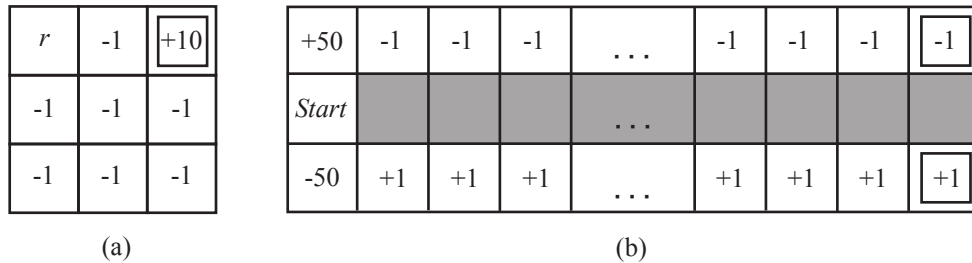


Figure 1. (a)  $3 \times 3$  world for ???. The reward for each state is indicated. The upper right square is a terminal state. (b)  $101 \times 3$  world for ???. The start state has reward 0.

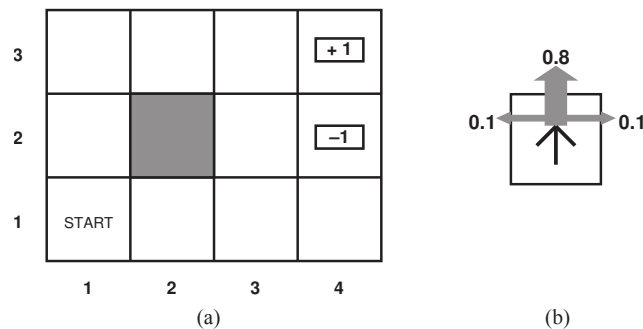


Figure 2. (a) A simple  $4 \times 3$  environment that presents the agent with a sequential decision problem. The two terminal states have reward  $+1$  and  $-1$ , respectively, and all other states have a reward of  $-0.04$ . (b) Illustration of the transition model of the environment: the “intended” outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement.

Implement value iteration for this world for each value of  $r$  below. Use discounted rewards with a discount factor  $\gamma = 0.99$ . Show the policy obtained in each case. Explain intuitively why the value of  $r$  leads to each policy.

1.  $r = -100$
2.  $r = -3$
3.  $r = 0$
4.  $r = +3$

## Exercise 10 (AIMA, Ex 17.9)

Consider the  $101 \times 3$  world shown in Figure ?? . In the start state the agent has a choice of two deterministic actions, **up** or **down**, but in the other states the agent has one deterministic action, **right**. Assuming a discounted reward function, compute the utility of each action as a function of  $\gamma$ . For what values of the discount factor  $\gamma$  should the agent choose **up** and for which **down**?

---

This simple example actually reflects many real-world situations in which one must weigh the value of an immediate action versus the potential long-term consequences, such as choosing to dump pollutants into a lake.

## Exercise 11 (AIMA, Ex 17.10)

Consider an undiscounted MDP having three states,  $(1, 2, 3)$ , with rewards  $-1, -2, 0$ , respectively. State 3 is a terminal state. In states 1 and 2 there are two possible actions:  $a$  and  $b$ . The transition model is as follows:

- In state 1, action  $a$  moves the agent to state 2 with probability 0.8 and makes the agent stay put with probability 0.2.
- In state 2, action  $a$  moves the agent to state 1 with probability 0.8 and makes the agent stay put with probability 0.2.
- In either state 1 or state 2, action  $b$  moves the agent to state 3 with probability 0.1 and makes the agent stay put with probability 0.9.

Answer the following questions:

1. What can be determined *qualitatively* about the optimal policy in states 1 and 2?
2. Apply policy iteration, showing each step in full, to determine the optimal policy and the values of states 1 and 2. Assume that the initial policy has action  $b$  in both states.
3. What happens to policy iteration if the initial policy has action  $a$  in both states? Does discounting help? Does the optimal policy depend on the discount factor?

## Exercise 12 (AIMA, Ex 17.19)

A Dutch auction is similar to an English auction, but rather than starting the bidding at a low price and increasing, the seller starts at a high price and gradually lowers the price until some buyer is willing to accept that price. If multiple bidders accept the price, one is arbitrarily chosen as the winner. More formally, the seller begins with a price  $p$  and gradually lowers  $p$  by increments of  $d$  until at least one buyer accepts the price.

Assuming all bidders act rationally, is it true that for arbitrarily small  $d$ , a Dutch auction will always result in the bidder with the highest value for the item obtaining the item? If so, show mathematically why. If not, explain how it may be possible for the bidder with highest value for the item not to obtain it.

### Exercise 13 (AIMA, Ex 17.21)

Teams in the National Hockey League historically received 2 points for winning a game and 0 for losing. If the game is tied, an overtime period is played; if nobody wins in overtime, the game is a tie and each team gets 1 point. But league officials felt that teams were playing too conservatively in overtime (to avoid a loss), and it would be more exciting if overtime produced a winner. So in 1999 the officials experimented in mechanism design: the rules were changed, giving a team that loses in overtime 1 point, not 0. It is still 2 points for a win and 1 for a tie.

1. Was hockey a zero-sum game before the rule change? After?
2. Suppose that at a certain time  $t$  in a game, the home team has probability  $p$  of winning in regulation time, probability  $0.78 - p$  of losing, and probability 0.22 of going into overtime, where they have probability  $q$  of winning,  $0.9 - q$  of losing, and 0.1 of tying. Give equations for the expected value for the home and visiting teams.
3. Imagine that it were legal and ethical for the two teams to enter into a pact where they agree that they will skate to a tie in regulation time, and then both try in earnest to win in overtime. Under what conditions, in terms of  $p$  and  $q$ , would it be rational for both teams to agree to this pact?
4. Some experts report that since the rule change, the percentage of games with a winner in overtime went up 18.2%, as desired, but the percentage of overtime games also went up 3.6%. What does that suggest about possible collusion or conservative play after the rule change?

## Quiz

Which of the following is true? In Markov Decision Processes, ...

- the closer the discount factor  $\gamma$  to 0, the higher the utility of future rewards.
- the closer the discount factor  $\gamma$  to 0, the longer Value Iteration may take to converge.
- the closer the discount factor  $\gamma$  to 1, the greedier the optimal agent.
- the closer the discount factor  $\gamma$  to 1, the longer Value Iteration may take to converge.

Q-Learning ...

- is a model-based reinforcement learning algorithm.
- is an on-policy reinforcement learning algorithm.
- converges to an optimal policy, but only when acting optimally.
- requires a random exploration strategy to converge to an optimal policy.

Assume that we run  $\epsilon$ -greedy Q-learning until convergence. What is the optimal policy  $\pi^*$  we obtain for an arbitrary state  $s$ ?

- $\pi^*(s) = \arg \max_s V(s)$
- $\pi^*(s) = \begin{cases} \arg \max_a Q(s, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$
- $\pi^*(s) = \arg \max_a Q(s, a)$
- $\pi^*(s) = \arg \max_s Q(s, a)$

Let us consider a robot wandering around at the Montefiore Institute. Which of the following is true?

- The robot and its environment can be modeled as a partially observable MDP.
- The Kalman filter can be used for determining accurately its past trajectory in the building.
- A convolutional neural network would be too large to fit in the robot's memory.
- Value iteration (with an admissible heuristic) can be used for decoding the speech of its visitors.

The Bellman equations  $Q(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a')$  form a system of  $n$  non-linear equations with as many unknowns. Which of the following is true?

- $n$  is the size of the action space  $\mathcal{A}$ .
- $n$  is the size of the state space  $\mathcal{S}$ .
- $n$  is the size of the state-action space  $\mathcal{S} \times \mathcal{A}$ .
- $n$  is the sum of the sizes of the action and state spaces.

In DQN (Mnih et al, 2015), a reinforcement learning agent is trained to ...

- to classify images.
- to control a robot in a simulated environment.
- to play Atari video games.
- to play the game of Go.

In DQN (Mnih et al, 2015), the Q-table is approximated with ...

- a hash table.
- a transposition table.
- a linear regression model.
- a convolutional neural network.