DS-GA 3001.005
NYU Center for Data Science

Reinforcement Learning

Homework 02

Exercice 1 (20 points)

Markov Decision Processes and Bellman Equations:

- **1.1 (10 points)**: What is a Markov Decision Process, and what is the Markov property?
  Solution: An MDP is a mathematical idealization of goal-directed learning from interaction with an environment. Simulating an MDP produces a sequence of n tuples (trajectory) $(s_t, a_t, r_{t+1}, s_{t+1})_{t=0}^n = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_n)$. The MDP dynamics is fully characterized by the joint probability of each possible s_{t+1} and r_{t+1} as a function of the immediately preceding state and action, s_t and $a_t \Rightarrow p(s', r \mid s, a) = p(s_{t+1} = s', r_{t+1} = r \mid s_t = s, a_t = a)$.

  Markov property: The state includes all information from past agent–environment interactions that influence the future $\Rightarrow p(s, r \mid s_t, a_t) = p(s, r \mid H_t, a_t)$.

- **1.2 (5 points)**: Write $v_\pi(s)$ as a function of $q_\pi(s, a)$
  Solution: $v^\pi(s) = \sum_a \pi(a \mid s)$ and $q^\pi(s, a) = E(q^\pi(s, a))$

- **1.3 (5 points)**: Write $v_*(s)$ as a function of $q_*(s, a)$
  Solution: $v^*(s) = \max_\pi v^\pi(s) = \max_a q^*(s, a)$

Exercice 2 (30 points)

Dynamic Programming:

- **2.1 (20 points)**: In the gridworld problem below, the goal is to reach state g, the reward is -1 for moving to any state except state g where it is 0, actions in each state are up, down, right or left (by 1 step), and actions taking the agent off the grid leaves the state unchanged. What are the final state values after convergence of the *Value Iteration* algorithm?

		g	

Solution :

-4	-3	-2	-3
-3	-2	-1	-2
-2	-1	0	-1
-3	-2	-1	-2

We also accepted answers that took state value of 0 for states adjacent to G and so on

- **2.2 (5 points)**: What is the key difference between the Policy Iteration algorithm and the Value Iteration algorithm?
  Solution: VI is the same as PI except it truncates policy evaluation afterjust one loop (one update of each state). It is equivalent to turning the Bellman optimality equation into an update function

- **2.3 (5 points)**: What is the key difference between synchronous Value Iteration and asynchronous Value Iteration?
  Solution: Asynchronous DP backs up states in any order whatsover, while synchronous DP proceeds with an entire loop over all possible states at each iteration

Exercice 3 (25 points)

Monte Carlo and Temporal Difference:

- **3.1 (5 points)**: What is the key difference between Dynamic Programming and sample-based Reinforcement Learning?
  Solution: We can accept any valid answer. Below are a few options:

  Dynamic Programming requires a perfect model of state transitions and rewards to carry out a one-step look-ahead full-width backup at each iteration.

Sample-based RL uses a sample of experience to learn without a model. For example, instead of learning the true expected return $v^{\pi}(s) = E^{\pi}(G_t \mid s)$, we can sample its average:

$$v_{t+1}(s_t) = v_t(s_t) + \alpha_t \left(\sum_{k=0}^{T} \gamma^k r_{t+k+1} - v_t(s_t) \right)$$

- **3.2 (5 points)**: List at least three differences between Monte-Carlo policy evaluation and Temporal Difference policy evaluation
  Solution: We can accept any valid answer. Below are a few options:
  - MC must wait until the end of the episode before the return is known.
  - MC can only learn from complete sequences.
  - MC can only work for episodic (terminating) environments.
  - MC converges to the return $G_t = (r_{t+1} + \gamma(r_{t+2} + \dots))$ as an unbiased estimate of $v^{\pi}(s_t)$, but has high variance.
  - MC converges to the best mean-squared fit for observed returns.
  - MC does not bootstrap.

  - TD can learn after every step, before knowing the final outcome.
  - TD can learn from incomplete sequences.
  - TD can work in continuing (non-terminating) environments.
  - TD converges to the TD target $r_{t+1} + \gamma v(s_{t+1})$, which is a biased estimate of $v^{\pi}(s_t)$, but has lower variance.
  - TD converges to the solution of the maximum likelihood Markov model.
  - TD bootstraps.

- **3.3 (10 points)**: Why is Q-learning considered an off-policy control method? Provide an example where using an off-policy method is more appropriate than using an on-policy method.
  Solution: Because off-policy control means optimizing behavior by learning about a target policy from experience sampled by following a different policy called behaviour policy, and Q-learning systematically updates the greedy policy whatever the behavior policy followed is. If you want to explore a large action space efficiently, off-policy methods are better suited, for ex : Autonomous Driving

- **3.4 (5 points)**: Why is Q-learning overly optimistic? And how does Double Q-learning help address this issue?
  Solution: Q-learning uses the same Q-function to select greedy actions and to evaluate their values. This tends to over-select overestimated values and under-select underestimated values, perpetuating an upward bias. Double Q-learning uses two independent Q-functions to select vs. evaluate actions. It tends to reduce the upward bias of Q-learning estimation errors.

Exercice 4 (25 points)

Produce the code to do the following::

- **4.1 (10 points)**: Write a function that implements the ϵ-greedy action-selection strategy. It should return the action selected. The required input to this function should be listed as arguments and/or defined in the function's docstring.
  Solution: Two different examples of implementations are given inside lab 4
- **4.2 (15 points)**: Write a function that implements a q-value update for tabular Q-learning, given a reward r observed after sampling an action a in a state s. It should return the updated q-value. The required input to this function should be listed as arguments and/or defined in the function's docstring.
  Solution: Two different examples of implementations are given inside lab 4