Homework 5

Francesco Mandruzzato Mat: 1204532

1 Reinforcement Learning - Introduction

The theory behind reinforcement learning is directly linked with the human and animal behavior, in particular it studies how a software agent takes actions inside an environment in order to maximize its control of the environment itself. In real world tasks however, agents faces very complex situations and they must derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experience to new situations. Recent advancements in deep learning have developed new algorithms able to work successfully in very hard environments. Every action taken by the agent has a consequence which is measured through the reward R_t and reinforcement learning is based on the reward hypothesis.

At each time step t the agent executes an action A_t given an observation O_t of the environment, receiving R_t. It is possible to define an history $H_t = O_1, R_1, A_1, \ldots, A_{t-1}, O_t, R_t$ of what happened in the past. A state S_t is a function of the history such that $S_t = f(H_t)$ and it is used in order to determine what happens next. Two different states are present, the environment state (which is not always visible by the agent) and the agent state itself. Given a generic state S_t we call it a Markov state iff $P[S_{t+1}|S_t] = P[S_{t+1}|S_1, S_2, \ldots, S_t]$ which means that the future state can be derived given only the present state S_t.

An agent has a policy which describes its behavior, in particular it defines a map from the state space to the action space where in the more general case its a stochastic process defined as $\pi = P(a|s) = P[A_t = a|S_t = s]$. We define the action-value function for policy π denoted as q_π, which tells us how good it is for the agent to take any given action from a given state while following policy π. In other words, it gives us the value of an action under π, and is given by:

$$\begin{aligned} q_\pi(s,a) = E_\pi[R_{t+1} + \lambda R_{t+2} + \lambda^2 R_{t+3} + \ldots |S_t = s, A_t = a] &= E_\pi[G_t|S_t = s, A_t = a] \\ &= E_\pi[R_{t+1} + \gamma \sum_{k=0}^{\infty} \lambda^k R_{t+k+2}|S_t = s, A_t = a]. \end{aligned}$$

Conventionally, the action-value function q_π is referred to as the Q-function, and the output from the function for any given state-action pair is called a Q-value. The letter 'Q' is used to represent the quality of taking a given action in a given state.

The goal of RL algorithms is find the policy which maximize the cumulative rewards for the agent, hence the optimal action-value function given by the optimal policy as:

$$q^*(s,a) = \max_{\pi} q_{\pi}(s,a), \tag{1}$$

for all $s \in S$ and $a \in A$. In other words, $q*$ gives the largest expected return achievable by any policy π for each possible state-action pair. If we look at the previous equation we can derive the so-called Bellman equation for the Q-function:

$$q^*(s,a) = E[R_{t+1} + \lambda \max_{a'} q^*(s',a')] \tag{2}$$

This is called the Bellman optimality equation. It states that, for any state-action pair (s,a) at time t, the expected return from starting in state s, selecting action a and following the optimal policy thereafter is going to be the expected reward we get from taking action a in state s, which is R_{t+1}, plus the maximum expected discounted return that can be achieved from any possible next state-action pair (s',a').

1.1 Exploration vs Exploitation

During the learning a RL algorithm need to estimate the optimal policy or value function. In the training phase the algorithm faces the so called *exploration and exploitation* dilemma. *Exploration* is the selection and execution of an action which is likely not optimal according with the notion of the agent while *exploitation* is the selection and execution of an action which is optimal accordingly to the agent's knowledge. In RL we need to establish a trade-off between the two and one technique is the $\epsilon - greedy$ strategy. It defines an ϵ value which is in the [0,1] range for balancing exploitation and exploration respectively. As the agent acquires knowledge about the environment, the ϵ start decreasing and the agent makes use of the acquired knowledge to exploit the ambient. To decide the strategy of the agent we randomly pick a number between [0,1] and if the number is greater than ϵ the agent will exploit the environment and viceversa. Given the definition of $\epsilon - greedy$ we can define a policy π to be $\epsilon - greedy$. The problem with $\epsilon - greedy$ is that, when it chooses the random actions, it chooses them uniformly (i.e. it considers all actions equally good), even though certain actions are better than others. Of course, this approach is not ideal in the case certain actions are extremely worse than others. Therefore, a natural solution to this problem is to select the random actions with probabilities proportional to their current values. These policies are called softmax policies.

1.2 On-policy algorithms, SARSA

The first step is to learn an action-value function $Q(s,a)$. An on-policy algorithm needs to estimate $Q_{\pi}(s,a)$ for a current policy π and for every state s and action a. This is done by using the theory of Temporal Difference resulting in the following equation:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \lambda Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{3}$$

which define the **SARSA** algorithm, since the components $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$ are taken into account. The α value defines the learning rate while *lambda* is the discount factor.

1.3 Off-policy algorithms, Q-learning

An on-policy algorithm is an algorithm that, during training, chooses actions using a policy that is derived from the current estimate of the optimal policy. One of the most important algorithm is **Q-learning** defined by the following equation:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \lambda \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]. \tag{4}$$

The main difference between **SARSA** and **Q-learning** is in the updating rule. while **SARSA** updates the Q-table always relying on the following policy π which can be $\epsilon - greedy$, **Q-learning** updates the Q-table using a greedy policy (e.g $\epsilon = 0$). However, when actually taking an action, Q-learning still uses the action taken from a $\epsilon - greedy$ policy or one other type of policy.

2 Problem statement

In this homework we analyze the **SARSA** and **Q-learning** algorithms applied on a $10x10$ grid where an agent is moving trying to reach the goal. In particular different parameters are tuned for both algorithms (**episode-length**,**lambda**,**alpha**,**epsilon**) searching for the best combinations and providing a comparison between **SARSA** and **Q-learning**. Finally different obstacles are fed into the environment in order to better analyze the agent behavior.

3 Results

For the following tests a Grid Search has been implemented in order to find the better configurations of parameters. The best configuration is found as the best average reward obtained across the episodes. The agents can take 5 different actions (Up,Down,Left,Right,Stay).

Parameters	Values
alpha	[0.9, 0.8, 0.7, 0.5, 0.4]
discount	[0.99, 0.8, 0.7, 0.6, 0.5]
epsilon	[0.9,0.8,0.7,0.6, 0.5, 0.4, 0.3]
episodelength	[100, 50]

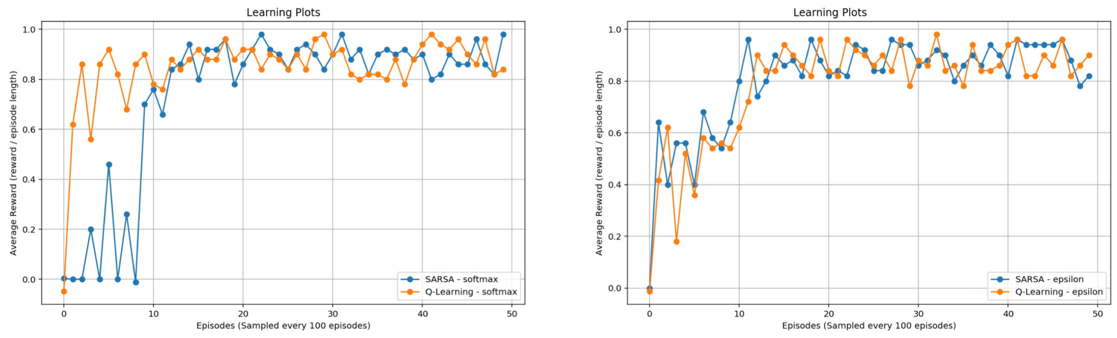

Figure 1: Average reward for SARSA and Q-Learning algorithms. The average reward plot has been sampled every 100 episodes in order to better visualize the results.

.

3.0.1 No obstacles environment

As previously introduced, we analyze the agent behaviour in a $10x10$ grid where the goal is fixed in the space with a reward equal to 1, all other cells have no reward and when the agent tries to go out of the grid it gets a reward of -1. In what follows **SARSA** and **Q-learning** have been tested with different parameters looking at the differences between the *softmax* and the $\epsilon - greedy$ policies. The principal difference is between **SARSA** with Softmax policy and $\epsilon - greedy$ one, in particular as expected with the epsilon-greedy policy **SARSA** seems to be more accurate, which is because epsilon-greedy policy is more conservative, especially in the first stages. One other interesting difference is between the **Q-learning** algorithm with the two mentioned polices. **Q-learning** tends to converge faster because it adopts a greedy policy and probably using an $\epsilon - greedy$ policy decrease its performances during the first stages. In general the main difference is given by the polices and not by the algorithms themselves.

3.0.2 Obstacles environment

For this experiment, different obstacles and are introduced. Walls are placed in the grid which gave a negative reward of -0.1 and they cannot be passed. Sand damage the robot circuits giving him a reward of -0.3, while going outside of the grid results in a negative reward of -0.6. The results achieved with **SARSA** and **Q-learning** are reported in the Fig.2. The agent can't start from a cell occupied by the goal or by a wall.

From the results obtained it is possible to see that the agent is able to avoid the obstacles and reaching the goal. This time, the difference between the two polices is evident and it can be justified by the polices themselves. As previously explained with $\epsilon - greedy$ policy the agent is more conservative in the process of learning and this is probably because in the initial steps the Q-table has no a big

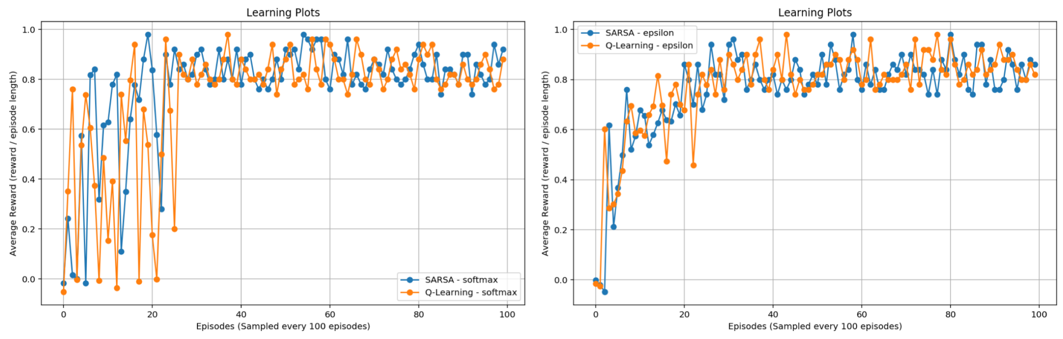

Figure 2: Average reward for SARSA and Q-Learning algorithms when some obstacles are introduced. The average reward plot has been sampled every 100 episodes in order to better visualize the results.

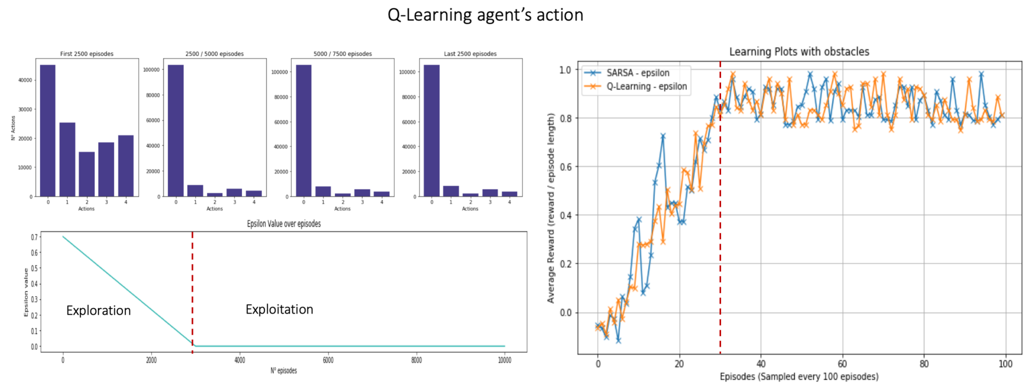

Figure 3: Agent behaviour with **Q-Learning** algorithm and modified version of *epsilon-greedy* policy. As it is possible to see, during the exploration phase the agent moves towards the grid studying the environment, after the exploration phase, it starts exploiting achieving max reward. In the learning plot, also **SARSA** performance is shown. The action are the following: *Stay*:0, *Up*:1, *Down*:2, *Left*:3, *Right*:4

.

distinction between the best actions and the worst ones. Hence, depending on the application one policy is preferred w.r.t the other.

3.1 Further considerations

The results obtained in this experiment and the agent behaviour reported in Fig.3 drive us to the last experiment of this Homework. In this experiment a modified version of the $\epsilon - greedy$ policy is introduced, where in the first episodes the epsilon start decreasing and after 3000 episodes it is set to zero.

Params	SARSA (ϵ)	SARSA $softmax$	Q-Learning (ϵ)	Q-Learning $softmax$
$alpha$	0.6	0.5	0.5	0.5
$discount$	0.7	0.7	0.6	0.5
$epsilon$	0.4	0.3 (temperature)	0.4	0.3 (temperature)
$episodelength$	50	50	50	50

This is done because it is interesting to create a net distinction between the exploration and the exploitation phase. From the results obtained it is possible to notice that **SARSA** appears to be a little bit more conservative in the initial phase following its policy. An algorithm like **SARSA** is typically preferable in situations where we care about the agent's performance during the process of learning / generating experience. If our agent is an expensive robots which need to go on Mars, it will break if it falls down a cliff meaning that we'd rather not have it fall down too often during the learning process. However, we also know that we need it to act randomly sometimes. This means that it is highly dangerous for the robot to be walking alongside the cliff, because it may decide to act randomly (with probability epsilon) and fall down. So, we'd prefer it to quickly learn that it's dangerous to be close to the cliff.

The **Q-Learning** algorithm instead does not use the behaviour policy to select an additional action, it estimates the expected future returns in the update rule following a greedy policy. An algorithm like **Q-Learning** would be preferable in situations where we do not care about the agent's performance during the training process, but we just want it to learn an optimal greedy policy that we'll switch to eventually and this is visible from the Fig.3 in the intial phase. In conclusion, it is possible to say that both algorithms achieve the task of reaching the goal and depending on the type of application one algorithm may be preferred with respect to the other. In conclusion the figure of the grid is reported with two real example of the agent path which has learned during the process of learning. It is possible to see that even if the sand cells lead to some penalty, the agent still traverses them since allows a faster reaching of the target.

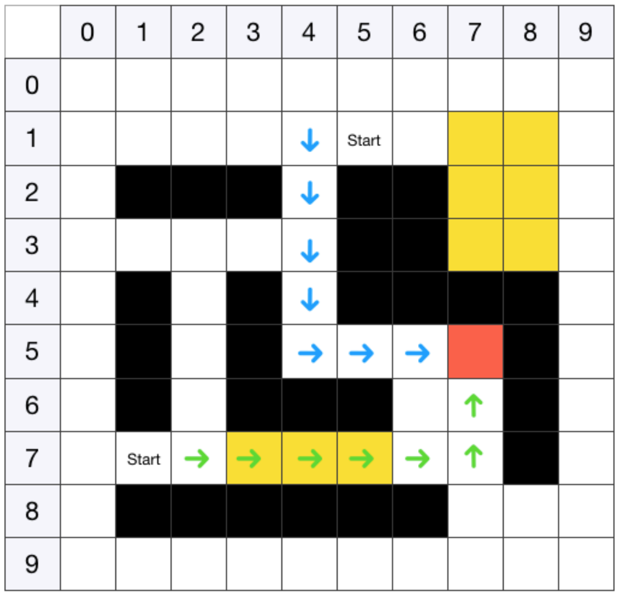

Figure 4: Two different simulations of the agent which has learned how to reach the goal avoiding to hit the walls