Model-free Reinforcement Learning: Q-Learning

Course assignment: Machine Learning
University of Piraeus, Demokritos

Sarafidis, Tasos
sar.tasos@gmail.com

Papadopoulos, Vasileios
vassilispapadop@gmail.com

March 7, 2021

1 Reinforcement Learning

Reinforcement Learning is an area of Machine Learning, where the main purpose is to find a way for an agent to learn a policy by interacting with its environment. An agent lives within an environment and interacts with it by examining the state in which it is in, for every time step. After examining its state, the agent chooses an action and the environment returns the corresponding reward, as well as the agent's new state. The agent's goal is to find the actions that return the most reward by trying them, without knowing which actions to choose beforehand. This means, the agent must discover the best action that will allow it to learn the optimal policy towards achieving its goals.

In Reinforcement Learning problems, the agent is always in interaction with the environment. In every time step t, the agent receives a state of the environment, st, and chooses an action, at. In the next time step, the agent gets a reward, r_{t+1}, from the environment and moves to the next state, s_{t+1}. In every step, the agent assigns a state to an action thus forming the agent's policy, π_t.

Problems in the area of Reinforcement Learning are usually modeled as Markov Decision Processes (MDPs). A MDP is a time-discrete stochastic control process, where agents' actions affect immediate and future rewards, as well as next states. A Markov Decision Process can be described by a tuple S,A,R,T, where:

- S is the set of all possible states.
- A is the set of actions.
- R(s) is the reward function, which returns a numerical value as a result of an agent's action.
- $T(s, \alpha, s')$ is the state transition function, which given a state s and an action α returns the next state s'.

2 Multi-Agent Systems

A Multi-Agent System is a set of agents who interact with each other and have a common or contradictory goal. In multi-agent systems, each agent is a part of the other agents' environment, meaning that as an agent learns and looks for the best policy, the results of its actions depend, not only on its state, but also on the other agents' actions. Agents either cooperate with each other to achieve a common goal, or come into conflict with each other in order to succeed their personal goals.

The Markov Decision Process that was described in section 1, can be also expanded on multi-agent systems and can be denoted by a tuple (Ag, S,A,R,T):

- Ag is set of agents.
- S is set of states.
- A is set of possible actions.
- R is the reward function.
- T is the transition function.

3 Q-Learning

Q-Learning is an off-policy reinforcement learning algorithm that allows an agent to choose the best action in a given state. When the agent is in a state

and performs an action, the environment returns the reward and the agent's next state. Then, the agent estimates the value of the new state. Each time an agent chooses an action and learns about its new state, the overall value of the executed action in the corresponding state is updated by the *update rule*:

$$Q(s,a) = Q(s,a) + a[R(s,a) + \gamma \max Q(s',a') - Q(s,a)] \quad (1)$$

where $0 < \alpha \leq 1$ is the learning rate, $0 \leq \gamma \leq 1$ is the discount rate which determines the values of future rewards for a given time, and s' is the agent's next state after performing its action. The overall value is called Q-value and is stored in the Q-table, which is later being used by the agent to read the stored values.

There are two ways for an agent to choose its action in each state. Firstly, the agent chooses randomly its next action without looking up the Q-table. This method is called *exploration*. The second way of choosing its action is called *exploitation*. According to this, the agent looks up the Q-table to find out which action has the best Q-value and execute it.

But, how does the agent decide whether to explore or exploit? This is done by using the $\epsilon - greedy$ algorithm. This algorithm states that with a probability of $0 \leq \epsilon \leq 1$ an agent chooses to explore and with a probability of the agent chooses to exploit the Q-table.

4 Problem Formulation

In our assignment we were faced with a symmetrical game of cooperation between two agents, the agent *Row* and the agent *Column*. Each agent interacts with the environment, which in return gives the agents their reward, based on the combination of both their actions (Table 1).

	A	D
A	α, β	β, α
D	α, β	δ, ϵ

Table 1: Reward Matrix

According to Table 1, if both agents choose action A then agent *Row* will get reward α and agent *Column* will get reward β. Equally, if both agents choose action D, then *Row* will get reward β and *Column* will get reward α. On the other hand, if Row performs action A and Column chooses action D, then their rewards will be δ, ϵ respectively. Finally, if Row decides to choose action D and Column action A, their respective rewards will be ϵ, δ. It must be noted that the agents do not have any knowledge about the other agent's actions and their goal is to choose the action that maximizes their personal expected payoff.

The purpose of our assignment is to develop a Q-Learning ϵ-greedy algorithm which learns to play the game and satisfy the agents goals. To achieve that, after developing the algorithm, our two-agents system runs 20 rounds per episode, for a total of 100 episodes. In each round, agents choose their actions and the environment returns a reward to each one of them. At this point, the round is finished, the Q-values of each agent are computed and they are updated in the Q-table. After running 20 rounds, the episode ends and a new episode starts from the beginning resetting everything except for the Q-table. This process is repeated until 100 episodes are completed.

5 Implementation

In our implementation we used the programming language *Python* and *numpy*, *pandas* and *matplotlib* packages. To represent the problem, we make use of pandas *DataFrame* object. We created the class *Agent* to describe the contents of the entity *agent* in the context of MDP framework. In particular, we let the agent to choose its *next-action*, discover environment states, update its q-table and store statistics. The function *next-action* implements the $\epsilon - greedy$ strategy to explore different states it could encounter. Implementation is shown below.

```
if np.random.random() < self.e-greedy:
    return np.random.choice(actions)
else:
    np.argmax(self.q-table[self.current-state])
```

As mentioned above, our approach lets the agent to discover states without explicitly defining the size of the q-table. Thus, we make an extra step every time *updateQ* function is called which is shown in the first two line of code-snippet below.

```
if (action, reward) not in self.q-table:
```

self.q-table[(action, reward)] = [0, 0]

old-value = self.q-table[self.curren-state][action]
next-max = np.max(self.q-table[self.current-state])

new-value = (1 - learning-rate) * old-value + learning-rate * (reward + discount-factor * next-max)

self.q-table[self.current-state][action] = new-value
self.current-state = (action,reward)

After having defined the behaviour of the class agent, the set up of the simulation is the following. We iterate for every episode and every game run and we let two identical agents to pick their next action. The environment *gives* back the respective reward to each one of them and they update their q-tables. We repeat for 150 episodes and 20 game runs per episode. At the beginning of the episode we set cumulative reward per agent, collected previously to zero. Furthermore, before starting the next game run(20 games) we decrement the variable ϵ-greedy by *0.01*; thus at the end of 100th episode the exploration stops and the agent starts to *trust* its q-table(*exploitation*).

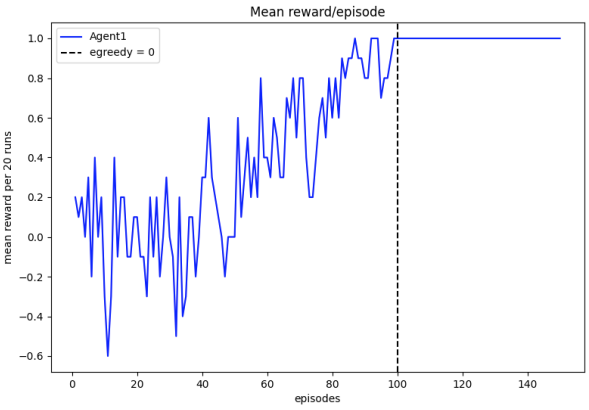

Figure 1: Average reward per episode for agent 1

Figures 1 and 2, depict the average reward both agents get at the completation of each episode. The vertical dashed line indicates that the exploration has stopped. As expected the agents receive the maximum possible reward after training which remains constant until the end of the remaining episodes.

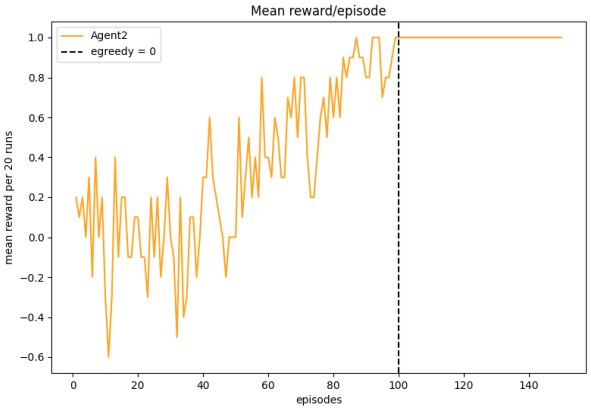

Figure 2: Average reward per episode for agent 2

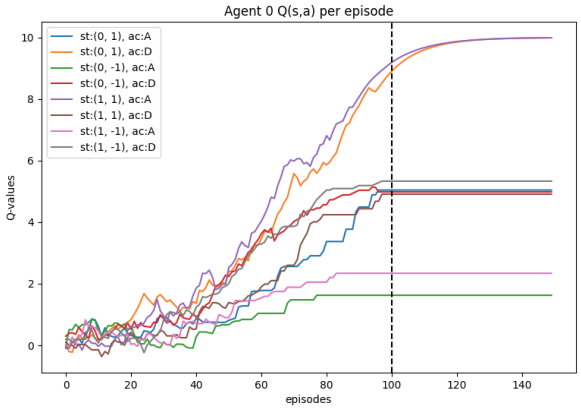

Figure 3: Q-values evolution per episode

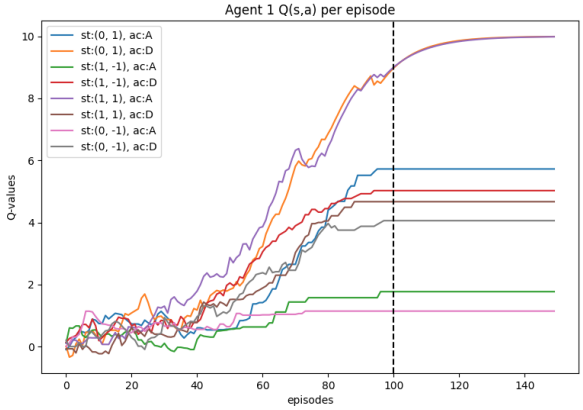

Figure 4: Q-values evolution per episode

Next, we show the evolution of each discovered state/value combination for the 2 agents; figures 2 and 3. It is worth mentioning that our agents perceive the notion of a state as the tuple of (action,reward). Thus, Q-values are represented as Q(Action,Reward,Next-action). It seems the agents learned that 2 particular q-values would lead to maximization of total reward.

Finally, we show the cumulative reward per agent during the whole simulation. Total time is defined as the product of *number-of-episodes* times *number-of-games*. At the time unit 2000, exploration stops and the reward increases linearly. This was hinted by figures 1 and 2.

As in Machine Learning in general, Reinforcement Learning has the so called *hyper-parameters*. In this simulation we set the learning-rate α to 0.1 and the discount-factor γ to 0.9 in order to make Q-values to converge. It is obvious the convergence depends also on the number of executed episodes. Lastly, it is possible for the two agents to reach Nash-Equilibrium **before** Q-values have converged.

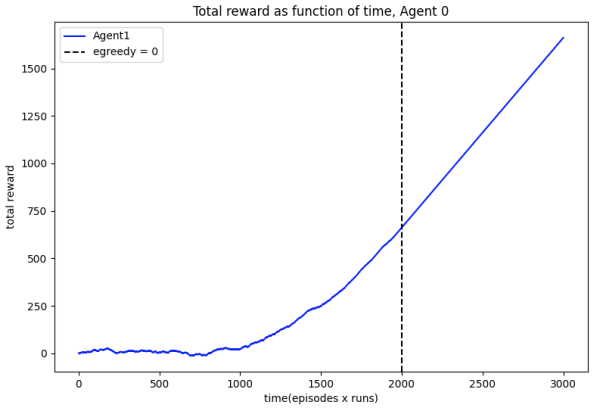

Figure 5: Total reward agent 1

6 Disclaimer

In order for our implemantion to be as generic as possible, we made the agents to *discover* states. This means that we did not hard-coded the size of Q-table even though it was a trivial calculation. Furthermore, we tried other two approaches which led to similar results. One approach was to model the state space as the number of games(20 in this case) while the other one, since it is a stochastic game state space can be thought as a single state[3].

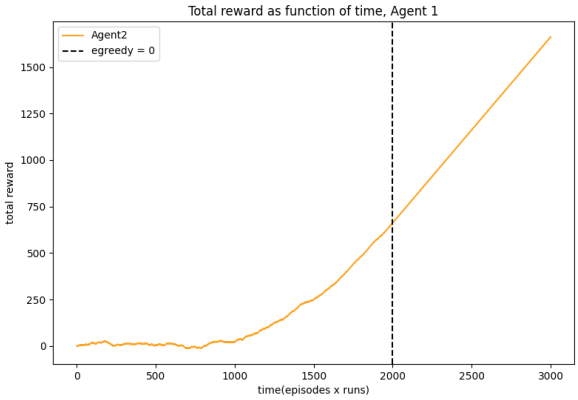

Figure 6: Total reward agent 2

References

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Fourth Edition 2020.

[2] Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction*. MIT Press, 2017.

[3] Junling Hu and Michael P. Wellman. *Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm*. University of Michigan, Ann Arbor, MI 48109-2110, USA.