Reinforcment Learning: Q-Learning Epsilon-Greedy algorithm application

Dimitrios Delikonstantis
University of Piraeus

Michael Zouros
University of Piraeus

February 27, 2021

Abstract

Reinforcement Learning constitutes one of the three basic Machine Learning paradigms, alongside Supervised Learning and Unsupervised Learning. Reinforcement Learning is concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward[1]. In this paper, we are studying the application of Q-Learning - a model-free Reinforcement Learning algorithm, combined with Epsilon-Greedy - a simple method for balancing an agent's exploration versus exploitation dilemma, on a Multi-Agent environment. Specifically, we consider a stochastic games approach framework, which consists of two identical agents (clones), which interact with their environment through two possible actions (alpha, delta) and receive one of the following rewards (a=1, b=1, d=-1, z=-1). Based on the study, we experiment on a number of episodes (150), rounds of each episode (20) and epsilon settings. The results suggest that, as time goes by, each agent chooses the best strategy (actions) for himself, which gives him the best payoff (reward), meaning that our multi-agent Q-Learning algorithm converges to a Nash equilibrium.

I. Introduction

Reinforcement Learning has gained extensive attention in the recent years. It is well suited for multi-agent systems, where agents know little about other agents, and the environment changes during learning. In this study, we are interested in the Nash equilibrium solution because we want to design learning agents for non-cooperative multi-agent systems. In such systems, every agent pursues its own goal and there is no communication among agents. Specifically in a Nash equilibrium, each agent's choice is the best response to the other agents' choices.
Our algorithm uses the q-learning and epsilon-greedy methods/techniques in order to achieve the convergence. It is designed for 2-player stochastic games, but it can also be extended to n-player games.

II. Methods

i. Problem Presentation

The script used in this work is implemented in Python 3.6 and the visualization of the graphs has been done with the help of matplotlib library. We define a symmetric, bi-matrix game between two agents of the following shape:

Table 1: *Actions and Rewards*

	Action A	Action D
Action A	a, b	d, z
Action D	z, d	b, a

whereas a=b=1, d=z=-1. The purpose of this study is to implement a Q-Learning, Epsilon-Greedy algorithm, which consists of an agent that learns to play the game with its clone. Each agent can either choose action A or action D. An agent does not know the action of the other agent. Each agent communicates with the environment, which returns the reward depending on the combination of actions of both agents. Additionally, each agent chooses an action independently of the other and individually pursues to maximize his own expected utility. For example, if row player agent chooses A and column player agent chooses D, the environment returns utility d to the row player and utility z to the column player.

ii. Q-Learning Epsilon-Greedy Algorithm

We run a number of 150 episodes where each episode consists of 20 game rounds. In each round, each agent chooses an action. When both agents complete their action they fall on a state. After that, the environment returns the reward of the state to each agent. We define 4 possible states for each combined action. The following table presents the corresponding states depending on the combined actions.

Table 2: *States*

	Action A	Action D
Action A	state 1	state 2
Action D	state 3	state 4

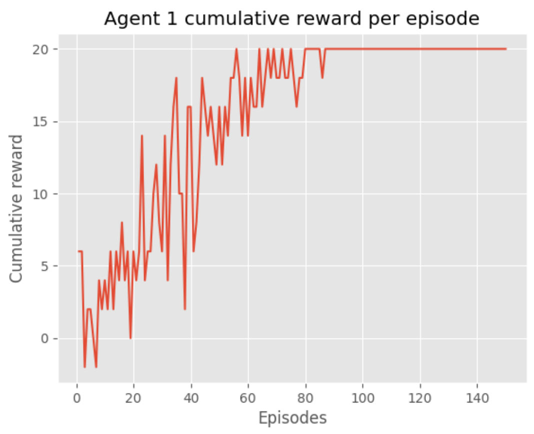
We start at episode 1 with epsilon=1 and decrease by 0.01 for each episode. By the time we reach episode 100 epsilon equals 0 and then we run 50 more episodes with epsilon=0 to observe the equilibrium state. Epsilon balances between exploration and exploitation. A factor epsilon=1, means that the agents act randomly and explore the environment. Instead of selecting actions based on the max future reward the agents select an action at random. This is called exploration. A factor epsilon=0 means that the agents select an action based on the max q-value from their respective Q-tables. This is called exploitation.
We set the discount factor gamma=0.9 throughout our experiment. A discount factor gamma=0.9 means that the agents give more importance to the future reward than the immediate reward. Also, we chose a learning rate factor alpha=0.6. This way, we balance the acceptance between the new values and the old values.
We defined a Q-table for each agent that follows the shape Q[state][action] and we initialize its values to 0. Q-tables are used to store (update) the q-values for each agent after each round. The Q-table is a reference table for each agent to select the best action based on its q-values.
On the start of each episode, time is set to 0. Then, the first round starts and each agent chooses an action. When both agents perform an action they fall on a state and receive the corresponding reward for this state. Then the Q-tables are updated and timing is increased by 1. Then the next round occurs.

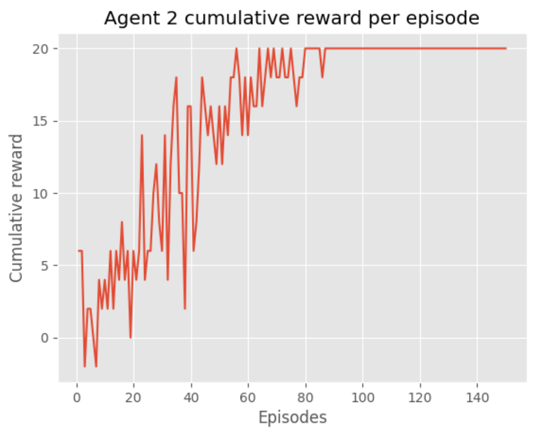
iii. Evaluation

The evaluation of our algorithm's performance has been done with respect to the cumulative reward each agent amass during the progress of the episodes. Specifically, the evaluation of our algorithm's performance takes place during the last 50 episodes, where the epsilon value equals to zero (meaning our agents will only exploit their respective Q-tables). It is much like the "traditional" machine learning algorithms, where we train and then validate our classifier/regressor. In our example, we trained our algorithm for 100 episodes and then tested its performance for the next 50 episodes. It is on these last 50 episodes that we test how well we have trained (updated) our agents' Q-tables, which involves all the information needed so our agents to do the best possible actions and gain the most profit.

III. Results

We present the results of our study in a series of graphs. In the first two graphs, we visualize the cumulative rewards for each agent for all 150 episodes (20 rounds each). We notice how, as time goes by, and after approximately 90 episodes, both agents start choosing the best for them actions, accumulating the max reward (+1 reward x 20 rounds = +20)



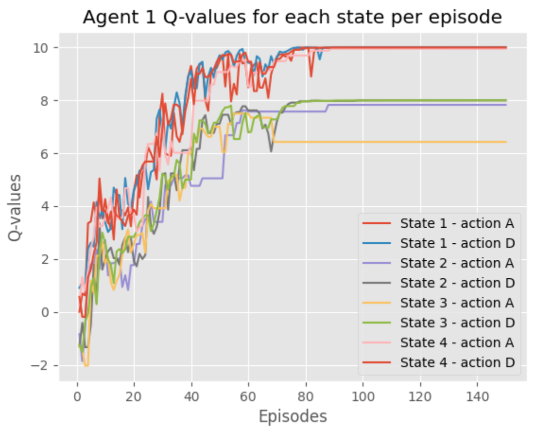


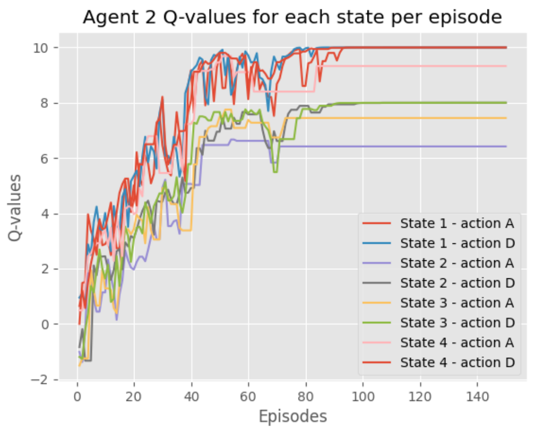
At this moment, it is important to clarify a few things:

- Our Q-Learning starts (hypothetically) at zero (0) time. We consider a timestep of +1 for measuring each round's duration. In the whole duration of our loop (150 episodes of 20 rounds each) this equals to 3000 timesteps. Plotting the graphs of the cumulative rewards for each agent per timestep proved visually messy/noisy. For this reason, we decided plotting the cumulative rewards for each agent per episode instead.

- In the "Problem Presentation" section, we declared four reward variables for our agents (a, b, d, z), with values a = b = 1 and d = z = -1. Observing Table 1, we can easily notice that this set of values will make both our agents get the same reward in each round. This is the reason why our cumulative rewards graphs look-alike. For this reason, it is important to clarify that the algorithm is generalized and we can set whatever values we want (as long as d, z < a, b) without needing to interfere with the code.

In the following two graphs, we visualize each agent's actions' value (Q-value) for the same duration (150 episodes, 20 rounds each).





From the above graphs, it is becoming clear that not each and every action has the same positive result. We notice how specific actions on specific states alter the Q-values over time (150 episodes, 20 rounds each), with bad actions (leading to bad rewards) having worse performance.

IV. Discussion

The results suggest that, as time goes by, each agent chooses the best strategy (actions) for himself, which gives him the best payoff (reward). This leads to the conclusion that our multi-agent Q-Learning algorithm converges to a Nash equilibrium.
In this work we implemented a Q-Learning Epsilon-Greedy algorithm in order to study a two agent bimatrix game. The study can be generalized on n-agents, with more actions and bigger reward scales. It can also be extended on other types of Reinforcement Learning implementation methods.

References

[1] Wikipedia the Free Encyclopedia, "Reinforcement Learning", at: `https://en.wikipedia.org/wiki/Reinforcement_learning` (Feb 2021)

[2] Hu, Junling, and Michael P. Wellman, "Multiagent reinforcement learning: theoretical framework and an algorithm", ICML. Vol. 98. 1998.

[3] Sutton, Richard S., and Andrew G. Barto, "Reinforcement learning: An introduction", MIT press, 2018.

[4] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8.3-4 (1992): 279-292.

[5] A. Violante, "Simple Reinforcement Learning: Q-learning", at: `https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56` (Mar 2019)

[6] C. Shyalika, "A Beginners Guide to Q-Learning", at `https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c` (Nov 2019)

[7] baeldung, "Epsilon-Greedy Q-learning", at `https://www.baeldung.com/cs/epsilon-greedy-q-learning` (Jan 2021)

[8] A. Parkinson, "The Epsilon-Greedy Algorithm for Reinforcement Learning", at `https://medium.com/analytics-vidhya/the-epsilon-greedy-algorithm-for-reinforcement-learn` (Dec 2019)

[9] P. P. Ippolito, "Game Theory in Artificial Intelligence", at `https://towardsdatascience.com/game-theory-in-artificial-intelligence-57a7937e1b88` (Sep 2019)

[10] M. Lanctot, "Multiagent Reinforcement Learning", at `https://rlss.inria.fr/files/2019/07/RLSS_Multiagent.pdf` (Jul 2019)

[11] S. Hawl, "Epsilon-Greedy Algorithm in Reinforcement Learning", at `https://www.geeksforgeeks.org/epsilon-greedy-algorithm-in-reinforcement-learning/` (May 2020)