Q-Learning Code Readme

Overview

The code implements the Q-learning algorithm for a grid of size (Nx X Ny). GridWorld is a 2D rectangular grid of size *(Ny, Nx)*. In the GridWorld an agent starts off at any “grid square” and try to move to another grid square located elsewhere in its world. The objective is to discover optimal paths and policies for agents on the grid to get to their desired goal “grid square” in the least number of moves.

Classes

Environment Class- The class implements the environment of the grid and consists of two functions **getActions** and **getRewards.** The **getActions** function returns the possible set of actions for a state i.e for example if in corner state only two possible actions. The **getRewards** function returns the reward received on transitioning a state.

Brain Class- The brain class implements the brain of the system it has initializes the Qtable and keeps track of the updation of the values. Also, we implement the max Q-value select action through a function of the brain class. It has functions **updateQvalue** and **makeTransitionMax** to carry out the above-mentioned functions.

Agent Class- The agent class implements the agent of the system. It has the function **makeTransitionRandom** which carries out a random transition based on the exploration factor.

Hyper-parameters chosen

1. Discount Parameter(gamma) - 0.9
2. Learing Rate (alpha) - 0.1
3. Exploration factor (epsilon) - 0.9 with a decay parameter of 0.999

Note.- These parameters can be altered to check different results of our system

Running of the code

You could initialize the size of the table, Number of episodes at the top of the code. For better convergence choose the number of episodes to be 100 x max(Nx, Ny). On running the code on a Python 3.7 interpreter the console prints the reward collected for every 100 episodes passed. In the end, the rewards converge to a max value due to convergence in the Q-table and highly decayed exploration factor (High exploitation).