ENPM690: Robot Learning - Final Project Report

Balancing CartPole with Q-Learning

-Rishabh Mukund (UID: 117556124)

1. Introduction:

A free rotating joint connects a pole to a cart that runs along a frictionless track. A force of +1 or -1 is applied to the cart to move it right or left respectively. The goal is to keep the pole from falling over. Reinforcement Learning is used to balance the pole. Every timestep that the pole remains erect earns you a +1 reward. When the pole is more than 15 degrees from vertical or the cart goes more than 2.4 units away from the center, the episode terminates. Each episode is 500 frames long so the maximum reward is 500. Using Q-Learning based on MDP we try to maximize the rewards.

2. Algorithm:

Q-learning is a model-free reinforcement learning algorithm for determining the worth of a certain action in a given state. It can handle problems with stochastic transitions and rewards without requiring adaptations and does not require a model of the environment (thus "model-free").

For any finite Markov decision process (FMDP), *Q*-learning finds an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state. *Q*-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy. "Q" (Quality) refers to the function that the algorithm computes – the expected rewards for an action taken in a given state.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)}^{\text{temporal difference}}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

MDPs are intended to be a simple way of framing the problem of learning from interaction in order to accomplish a goal. The agent is the learner and decision maker. The environment is the item it interacts with, and it consists of everything outside the agent. The agent chooses actions, and the environment reacts to those actions by providing new situations to the agent.
1 The environment also creates rewards, which are particular numerical values that the agent tries to maximize over time by taking certain actions.

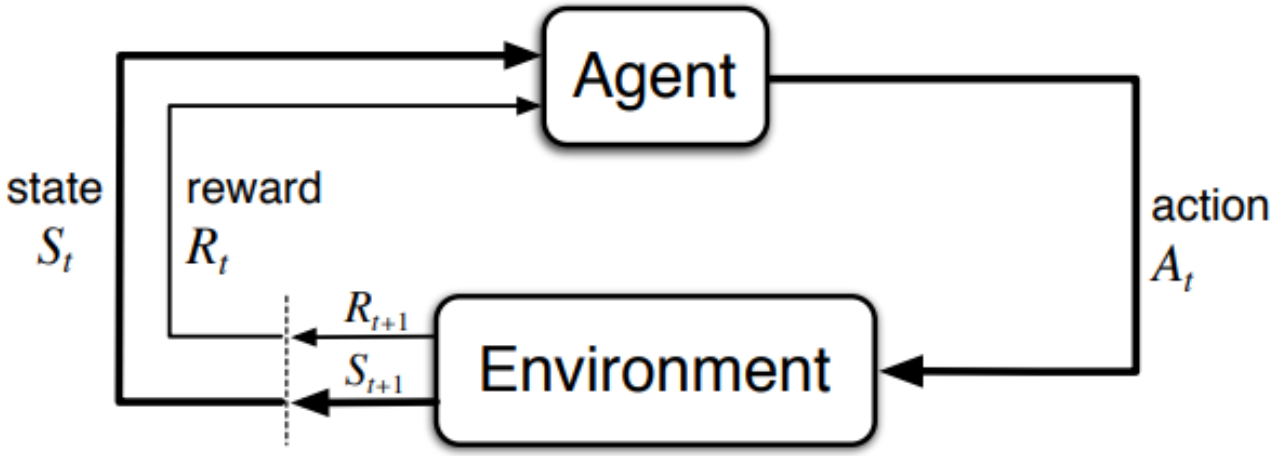


Fig: Agent Environment Interaction in MDP

The agent and environment interact at each of a sequence of discrete time steps, t = 0, 1, 2, 3,. At each time step t, the agent receives some representation of the environment's state, S[t] ϵ S, and on that basis selects an action, A[t] ϵ A(s). One time step later, in part as a consequence of its action, the agent receives a numerical reward, R[t+1] ϵ R, and finds itself in a new state, S[t+1].

3. Environment:

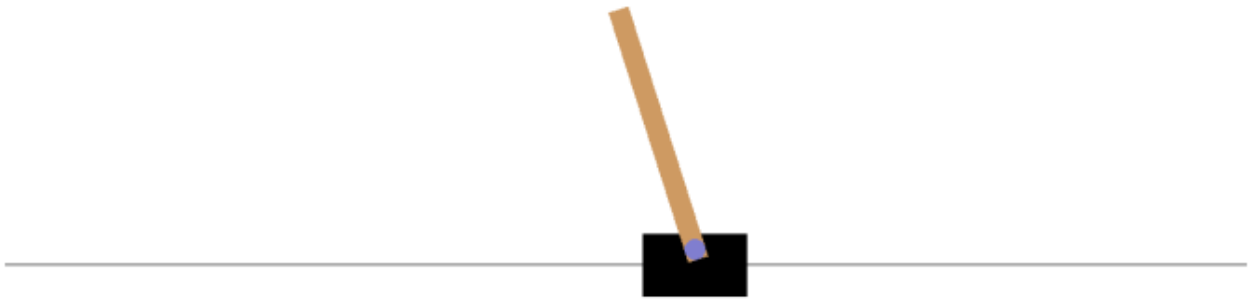


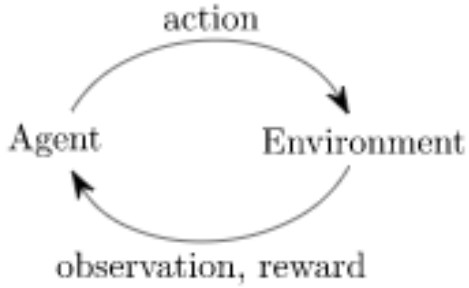
Fig: Environment of OpenAI's CartPole-v1

The environment's step() function returns exactly what we need. Step returns four values:

- observation (object): an environment-specific object representing your observation of the environment. For example, pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game.
- reward (float): amount of reward achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward.
- done (boolean): whether it's time to reset the environment again. Most (but not all) tasks are divided up into well-defined episodes, and being True indicates the episode has terminated. (For example, perhaps the pole tipped too far, or you lost your last life.)

- info (dict): diagnostic information useful for debugging. It can sometimes be useful for learning (for example, it might contain the raw probabilities behind the environment's last state change). However, official evaluations of your agent are not allowed to use this for learning.

This is just an implementation of the classic "agent-environment loop". Each timestep, the agent chooses an action, and the environment returns an observation and a reward.



The process gets started by calling reset(), which returns an initial observation.

4. **Pseudo Code:**

The following are the parameters utilized in the Q-value update process:

- *Learning Rate* (alpha): which can be set anywhere between 0 and 1. If you set it to 0, the Q-values will never be changed, and nothing will be learned. Setting a high value, such as 0.9, allows for rapid learning.
- *Discount Factor* (gamma): which can be set to anything between 0 and 1. The reality that future rewards are worth less than immediate rewards is modeled in this way. The discount factor must be less than 0 for the procedure to converge mathematically.
- Greatest Reward (maxα): that can be obtained in the next state after the current one. i.e. the reward for doing the best possible action after that.

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
   Initialize s
   Repeat (for each step of episode):
      Choose a from s using policy derived from Q
         (e.g., ε-greedy)
      Take action a, observe r, s'
      Q(s, a) <-- Q(s, a) + α [r + γ max_α,Q(s', a') - Q(s, a)]
      s <-- s';
   until s is terminal
```

Fig: QLearning Pseudocode

The above pseudocode explained:

1. Initialize the Q-values table, **Q(s, a)**.
2. Observe the current state, **s**.
3. Choose an action, **a**, for that state based on one of the action selection policies.
4. Take the action, and observe the reward, **r**, as well as the new state, **s'**.
5. Update the Q-value for the state using the observed reward and the maximum reward possible for the next state. The updating is done according to the formula and parameters described above.
6. Set the state to the new state, and repeat the process until a terminal state is reached.

5. References:

[1] https://gym.openai.com/docs/

[2] https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html

[3] https://deeplizard.com/learn/playlist/PLZbbT5o_s2xoWNVdDudn51XM8lOuZ_Njv