PROJECT 4 – TRAINING AN AGENT USING Q-LEARNING ALGORITHM

Varsha Ravichandiran
University at Buffalo
varshara@buffalo.edu

Abstract

Reinforcement learning is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, an agent faces a game-like situation and it employs trial and error to come up with a solution to the given problem. The agent gets either rewards or penalties for the actions performed. The goal is to maximize the total reward. Although the reward policies are set before, the agent knows nothing about the solution for solving the game. It's up to the agent to figure out how to perform the task to maximize the reward, starting from totally random trials and finishing with sophisticated tactics and superhuman skills. By leveraging the power of search and many trials, reinforcement learning is currently the most effective way to hint machine's creativity. In this project the agent is trained using Q-Learning algorithm.

1. INTRODUCTION

Q-Learning is a model-free reinforcement learning algorithm. The goal of Q-Learning is to learn a policy, which tells an agent what action to take under what circumstances. It does not require a model of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations. For any finite Markov decision process (FMDP), Q-Learning finds a policy that is optimal in the sense that it maximizes 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" names the function that returns the reward used to provide the reinforcement and can be said to stand for the "quality" of an action taken in each state. In this project, we will be implementing tabular Q-Learning, an approach which utilizes a table of Q-values as the agent's policy.

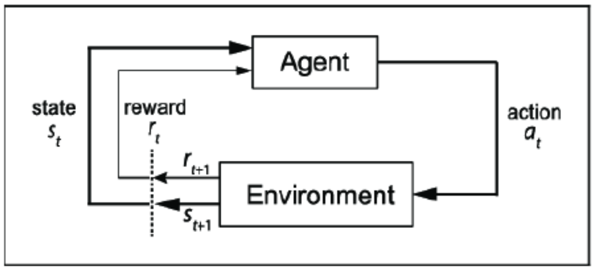


Fig 1.1: Reinforcement Learning Architecture

2. ENVIRONMENT

The environment provided for this project is a basic deterministic n * n grid-world environment (the initial state for a 4 * 4 grid-world is shown in Figure 2.1) where the agent (shown as the yellow square) has to reach the goal (shown as the green square) in the least amount of time steps possible. The environment's state space is described as an n * n matrix with real values on the interval [0, 1] to designate different features and their positions. The agent should work within an action space consisting of four actions: up, down, left, right. At each time step, the agent will take one action and move in the direction described by the action. The agent will receive a reward of +1 for moving closer to the goal and -1 for moving away or remaining the same distance from the goal.

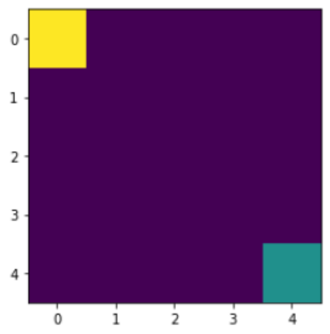


Fig 2.1: Initial state of Grid World Environment

3. ARCHITECTURE

Steps involved in training the agent using Q-Learning algorithm:

1. A grid world environment is defined using OpenAI's Gym library.

2. Hyperparameters used here are

- Alpha (learning rate): The alpha probability determines how much the agent values newly acquired information over the older data set. If alpha was 0 the agent would learn nothing new, and at a value of 1 would only make decisions based on the most recent data.
- Epsilon (exploration rate): To avoid getting stuck in a local minimum, the agent is made to explore. In this case, this means choosing a random action with the probability epsilon ($0 < \text{epsilon} < 1$).
- Gamma (discount factor): This rate determines how to preference rewards happening sooner rather than later. A discount factor of 0 would only consider the next reward; while 1 would give all future rewards equal weight. The goal is to keep the thing upright as long as possible, so the weight of all future rewards is equal.

3. Q-table is like brain of the agent and its values are defined for states and actions. Q (S, A) is an estimation of how good it is to take the action A at the state S. This estimation is iteratively computed using the TD- Update rule. The Q-table values are initialized to zeroes since the agent knows nothing about the problem solution.

4. An agent over the course of its lifetime starts from a start state, makes several transitions from its current state to a next state based on its choice of action and the environment the agent is

interacting in. At every step of transition, the agent from a state takes an action, observes a reward from the environment, and then transits to another state. If at any point of time the agent ends up in one of the terminating states that means, there are no further transition possible. This is said to be the completion of an episode.

5. The Temporal Difference or TD-Update rule can be represented as follows

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

   This update rule estimates the value of Q at every step of the agent's interaction with the environment. Hence the Q-Table is updated after every episode.

6. Choosing the Action to take using the greedy policy. It is a policy of choosing actions using the current Q-value estimations. It chooses the max of the future expected values of the given actions available.

7. Once an episode is finished, exploration rate is updated using exponential decay, which just means that the exploration rate decreases or decays at a rate proportional to its current value. We can decay the exploration rate using the formula as below

   epsilon = min_epsilon + (max_epsilon - min_epsilon) * np.exp (-delta_epsilon * ep)

   where delta_epsilon denotes the exploration decay rate.

8. Thus, after all the training steps we can see our agent performing well (it reaches the goal in the minimum possible steps).

4. RESULTS

The visualization graphs and the updated Q-Table are shown as below

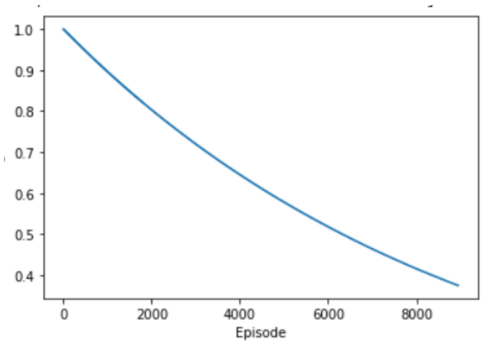


Fig 4.1: Epsilon VS Rewards

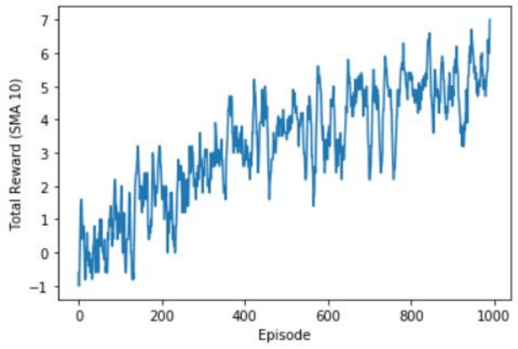


Fig 4.2: Rewards VS Episodes

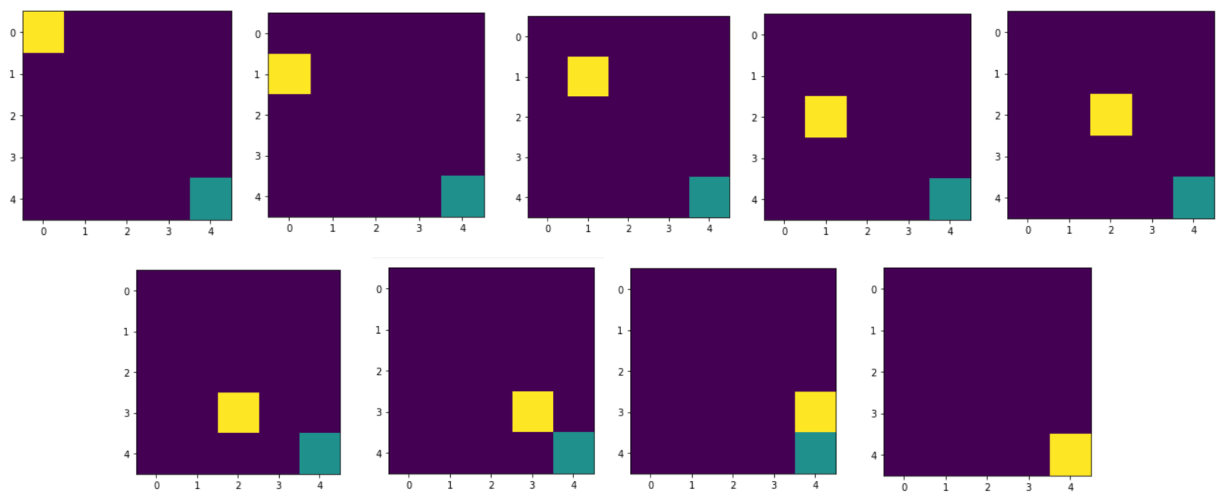
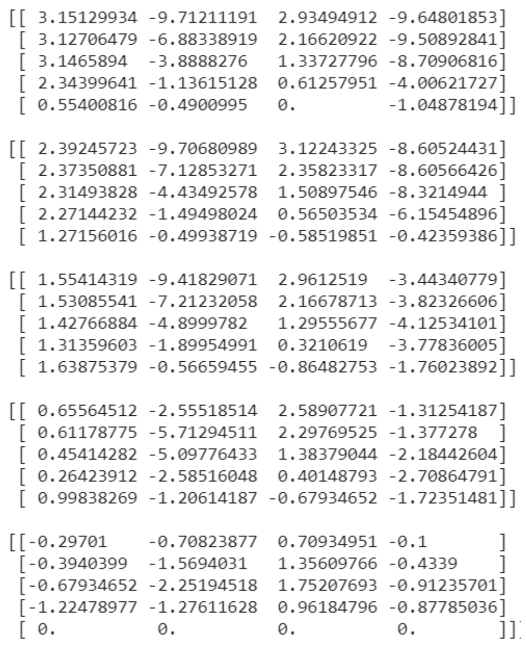


Fig 4.3: Grid World Agent Steps



```
[[ 3.15129934 -9.71211191  2.93494912 -9.64801853]
 [ 3.12706479 -6.88338919  2.16620922 -9.50892841]
 [ 3.1465894  -3.8888276   1.33727796 -8.70906816]
 [ 2.34399641 -1.13615128  0.61257951 -4.00621727]
 [ 0.55400816 -0.4900995   0.         -1.04878194]]

[[ 2.39245723 -9.70680989  3.12243325 -8.60524431]
 [ 2.37350881 -7.12853271  2.35823317 -8.60566426]
 [ 2.31493828 -4.43492578  1.50897546 -8.3214944 ]
 [ 2.27144232 -1.49498024  0.56503534 -6.15454896]
 [ 1.27156016 -0.49938719 -0.58519851 -0.42359386]]

[[ 1.55414319 -9.41829071  2.9612519  -3.44340779]
 [ 1.53085541 -7.21232058  2.16678713 -3.82326606]
 [ 1.42766884 -4.8999782   1.29555677 -4.12534101]
 [ 1.31359603 -1.89954991  0.3210619  -3.77836005]
 [ 1.63875379 -0.56659455 -0.86482753 -1.76023892]]

[[ 0.65564512 -2.55518514  2.58907721 -1.31254187]
 [ 0.61178775 -5.71294511  2.29769525 -1.377278  ]
 [ 0.45414282 -5.09776433  1.38379044 -2.18442604]
 [ 0.26423912 -2.58516048  0.40148793 -2.70864791]
 [ 0.99838269 -1.20614187 -0.67934652 -1.72351481]]

[[-0.29701    -0.70823877  0.70934951 -0.1       ]
 [-0.3940399  -1.5694031   1.35609766 -0.4339    ]
 [-0.67934652 -2.25194518  1.75207693 -0.91235701]
 [-1.22478977 -1.27611628  0.96184796 -0.87785036]
 [ 0.          0.          0.          0.        ]]]
```

Fig 4.4: Q-Table Values

5. CONCLUSION

This project helped me in getting a clear idea of how reinforcement learning is carried out in machine learning. It has also helped me to understand the functioning of Q-Learning algorithm. I was able to successfully train the agent and make it reach the goal in minimum number of steps. I was able to clearly understand the importance of the hyperparameters like exploration rate, alpha and gamma in training the agent. Thus, I was able to successfully perform Q-Learning approach in reinforcement learning on the grid world environment.

6. CHALLENGES

- It was difficult for me to understand the Q-Learning algorithm.
- I was not able to train my agent successfully because I implemented a wrong policy.
- I also found it difficult to update the exploration decay initially.
- Later, I implemented the correct steps and I was able to successfully train the agent to reach its goal in minimum number of steps

7. REFERENCES

1) Understanding the Q-Learning algorithm
https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe/
2) https://www.datahubbs.com/intro-to-q-learning/
3) https://medium.com/@_seanavery/reinforcement-learning-for-n00bs-pt-1-23ca71fc3c