Reinforcement Learning Based Game Playing Using Q Learning

Vineeth Thayanithi
University at Buffalo
`vthayani@buffalo.edu`

Abstract

Reinforcement learning is a learning strategy in which the machine is made to learn the actions it has to complete to reach a certain goal state. This method of machine learning is useful especially when we do not have data through which the machine can learn over. In this implementation, we build a model that can completes a simple game. The goal of the game is to reach a destination given a starting location. We use the Q learning algorithm for making the machine complete this game.

1 Introduction

1.1 Reinforcement Learning

Reinforcement learning is a sub branch of machine learning with supervised learning and unsupervised learning as its other major sub branches. Reinforcement learning aims at training a model to make a desirable set of actions. In most Reinforcement learning algorithms there exits the two most important elements of it, The Environment and The Agent. The Environment is the set of constraints that the agent is exposed to, generally it is expressed as the physical world in which the agent exists. An agent on the other hand is the component that decides the actions that has to be taken in a given situation.

The components Agent and Environment, together introduce a set of other components that are used for the given optimization problem namely state,action and reward. A state denotes the current situation of the environment, an action is the move that is performed by the agent on the environment which makes the state go into another state or stay at the same state. Each action that is performed on the environment comes with a reward associated along with it. The reward is what determines if the action performed causes a good or a bad effect on the state of the environment. When the given problem has been solved, the environment is said to have reached the goal state.

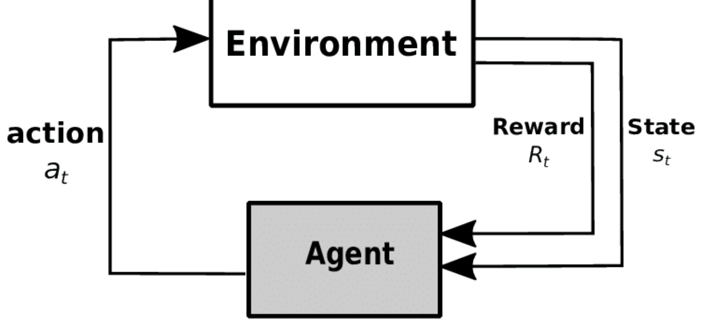


Figure 1: Architecture of Reinforcement Learning

The Agent performs the necessary actions on the environment by using what is called as the policy function. The policy function determines the best action to reach the next state given the current state. According to the policy function used by the learning process, policy functions can be categorised as on policy or off policy functions. An on policy agent learns the value based on its action from current policy whereas an off policy agent learns from its action based on another policy. We would be using an off policy strategy called as the q learning for our implementation.

1.2 Q Learning

Q learning also known as Quality Learning is an off policy reinforcement learning strategy where the goal is to the learn the policy π the maximizes V $^{\pi}$(s) for all states s.

where,

$$V^{\pi}(s) = r_t + \gamma r_{t+1} + \gamma 2 r_{t+2} + ... = \sum_{i=0}^{\infty} \gamma^i r_{i+1} \tag{1}$$

This kind of a policy is called as an optimal policy and is the argmax of V $^{\pi}$(s). However, it is difficult to learn such a function π^*. The optimal action for a given state is the one that maximizes the sum of the immediate reward and the value V^* of immediate successor state discontinue by γ.

$$\pi^*(s) = argmax[r(s,a) + \gamma V^*(\delta(s,a))] \tag{2}$$

1.2.1 Q Table

For Q learning, we create what is called as the q table. The Q table for our environment takes the shape [State, State, Action] where states denotes the current location of the box in the environment. We choose to initialize the q table with zeros and then update the q table with each iteration. Each Iteration in reinforcement learning is also called as an episode. For the learning process and updating the q table , the agent interacts with the environment in 2 different ways. The first method of learning is called as exploitation. In this method, the agent uses all possible combinations of state action pairs present in the q table. The agent selects the action which produces the greatest value. The second method, also known as exploration acts randomly. This allows the agent to discover new states which may be never be found during exploitation. Exploration avoids the agent from repeating actions that may not be the optimal action but yet is the best amongst all the given possibilities using exploitation. The Q table is updated after an action is performed on the environment and the updation ends when the goal state is reached. The basic rule for updating the q table is as below,

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{oldvalue} + \underbrace{\alpha}_{learningrate} \cdot \overbrace{\Big(\underbrace{r_t}_{reward} + \underbrace{\gamma}_{discountfactor} \underbrace{\max_a Q(s_{t+1}, a)}_{a} \Big)}^{learnedvalue} \tag{3}$$

α - Alpha is called as the learning rate which specifies how much of the new value is accepted over the old value.

γ - Gamma is the discount factor which is used for balancing the immediate rewards and the future rewards. The value of γ typically ranges from 0.8 to 0.99.

2 Environment

The environment for our project is deployed by using OpenAI Gym. OpenAI Gym is a developer toolkit for creating and comparing different reinforcement learning algorithms. Gym provides an open source interface to reinforcement learning tasks.

2.1 Grid Environment

The Grid Environment for the project is deployed using OpenAI Gym. The Grid Environment consists of a box in a two dimensional space of size 5x5 . The initial state of the environment starts with the box positioned at the top left corner of the grid environment. The terminal state of the environment is when the box ends at the bottom right corner of the grid space. The environment is as depicted below.

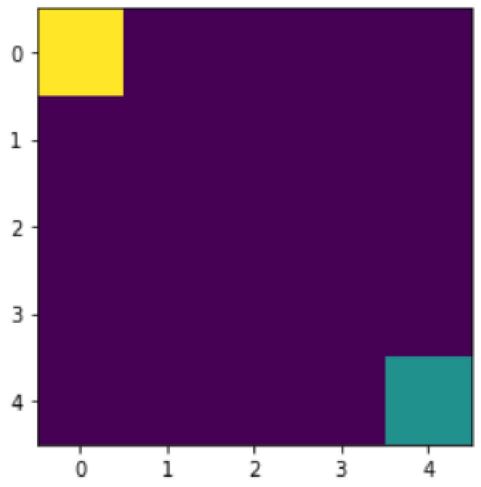


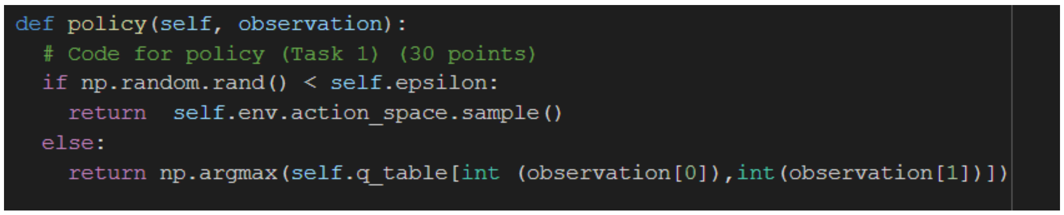
Figure 2: Environment of the RL Agent

The yellow block's position in the environment is the initial location and the yellow block moves around the 5x5 grid environment to reach the green box. The green box is the terminal state for which the agent needs to find routes to go to.

3 Implementation

3.1 Task 1

The task 1 for this problem is to write a policy function for the agent. The policy function determines whether the current iteration has to explore or exploit. As mentioned already, exploration says that the action is random whereas exploitation would use prior knowledge obtained from previous episodes. For coming up with such a policy function, we first generate a random value in the space (0,1) using numpy's random.rand() function. If the generated number is greater than the current epsilon value of the agent we would exploit otherwise, we would explore. The code snippet is as below.



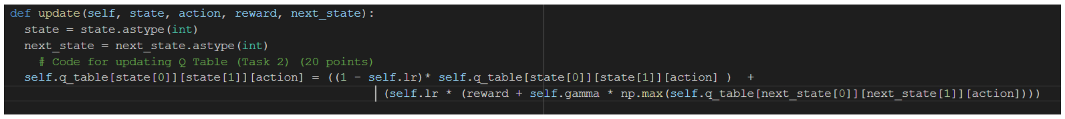
```
def policy(self, observation):
  # Code for policy (Task 1) (30 points)
  if np.random.rand() < self.epsilon:
    return  self.env.action_space.sample()
  else:
    return np.argmax(self.q_table[int (observation[0]),int(observation[1])])
```

Figure 3: Implementation of policy function

When the agent is not acting randomly, the agent will choose the action that produces the greatest reward for the current state.

3.2 Task 2

The second task is to update the q table. The updation of the q table is straightforward process. We use equation (3) for updating the q table. α determines how much the results from the rewards affect the old values. The code snippet for updating the q table is shown below.



```
def update(self, state, action, reward, next_state):
  state = state.astype(int)
  next_state = next_state.astype(int)
    # Code for updating Q Table (Task 2) (20 points)
  self.q_table[state[0]][state[1]][action] = ((1 - self.lr)* self.q_table[state[0]][state[1]][action] )  +
                                                    (self.lr * (reward + self.gamma * np.max(self.q_table[next_state[0]][next_state[1]][action])))
```

Figure 4: Implementation of policy function

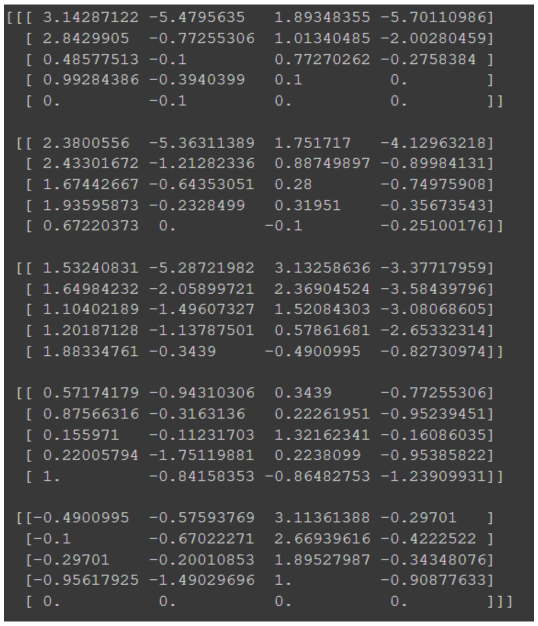
3.3 Task 3

The Final task is to train the agent to reach the terminal state. We train the agent for over 1000 episodes. The initial value of epsilon is 1 and a decay rate is set to 0.999. Hence, the epsilon value gradually decreases over each iteration thus eliminating the probability of randomness as the training proceed. The state is instantiated to the initial state for each episode, the variable done is set to False which tells if the terminal state has been reached and the current iteration reward is set to 0. While the variable done is still false, the action to be taken is found using the policy function. The next state, reward , done and info is obtained by the step function of the environment with the obtained action has its parameter. The q table is updated using the obtained reward. The new state is then set to the current state. We use exponential decay for updating epsilon. Exponential decay is updated as the max value between the minimum epsilon value and the product of the current epsilon value and the decay rate. Setting a minimum epsilon value would ensure that there is always some kind of randomness in the training process.

4 Results

4.1 Optimum Q table Values

After over 1000 episodes the optimum q table is as below.



```
[[[ 3.14287122 -5.4795635   1.89348355 -5.70110986]
  [ 2.8429905  -0.77255306  1.01340485 -2.00280459]
  [ 0.48577513 -0.1         0.77270262 -0.2758384 ]
  [ 0.99284386 -0.3940399   0.1         0.        ]
  [ 0.         -0.1         0.          0.        ]]

 [[ 2.3800556  -5.36311389  1.751717   -4.12963218]
  [ 2.43301672 -1.21282336  0.88749897 -0.89984131]
  [ 1.67442667 -0.64353051  0.28       -0.74975908]
  [ 1.93595873 -0.2328499   0.31951    -0.35673543]
  [ 0.67220373  0.         -0.1        -0.25100176]]

 [[ 1.53240831 -5.28721982  3.13258636 -3.37717959]
  [ 1.64984232 -2.05899721  2.36904524 -3.58439796]
  [ 1.10402189 -1.49607327  1.52084303 -3.08068605]
  [ 1.20187128 -1.13787501  0.57861681 -2.65332314]
  [ 1.88334761 -0.3439     -0.4900995  -0.82730974]]

 [[ 0.57174179 -0.94310306  0.3439     -0.77255306]
  [ 0.87566316 -0.3163136   0.22261951 -0.95239451]
  [ 0.155971   -0.11231703  1.32162341 -0.16086035]
  [ 0.22005794 -1.75119881  0.2238099  -0.95385822]
  [ 1.         -0.84158353 -0.86482753 -1.23909931]]

 [[-0.4900995  -0.57593769  3.11361388 -0.29701   ]
  [-0.1        -0.67022271  2.66939616 -0.4222522 ]
  [-0.29701    -0.20010853  1.89527987 -0.34348076]
  [-0.95617925 -1.49029696  1.         -0.90877633]
  [ 0.          0.          0.          0.        ]]]
```

Figure 5: Optimum Q Table

4.2 Graphs

4.2.1 Episodes vs Epsilon

The epsilon value for each episode is appended into an array and graph is plotted for episodes vs epsilon.

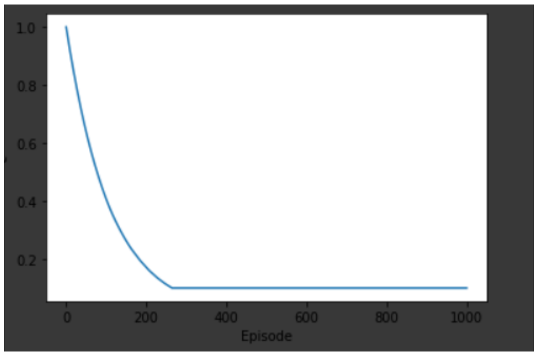


Figure 6: Episodes vs Epsilon

As we can see from the graph, the epsilon value gradually decreases and reaches the minimum value of 0.1.

4.2.2 Episodes vs Rewards

The reward for each action is obtained till the agent does not reach the terminal state. The sum of rewards for all actions is obtained for each episode and a graph is plotted for episodes against rewards.

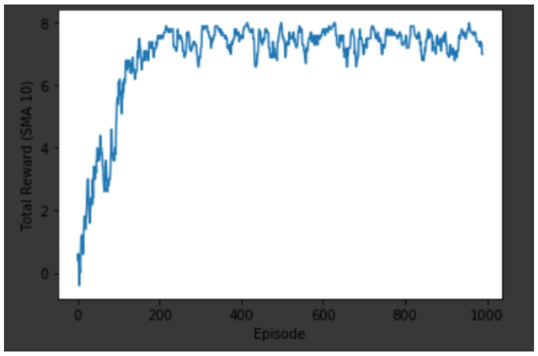


Figure 7: Episodes vs Epsilon

The fluctuations in the rewards is caused by the few randomness during each episode.

5 Conclusion

We have successfully trained the agent to reach the terminal state from the initial state. From the results, we observe that it takes around 8 steps for the yellow block to reach the terminal state.

References

[1] Reinforcement Learning: Overview, Professor Sargur N. Srihari

[2] Q Learning, Professor Sargur N. Srihari

[3] https://en.wikipedia.org/wiki/Q-learning

[4] https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

[5] https://www.geeksforgeeks.org/q-learning-in-python/

[6] https://stackoverflow.com/questions/48583396/q-learning-epsilon-greedy-update

[7] https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/