REINFORCEMENT LEARNING using OpenAI Gym

Uneeth Akula
50317480
Dept. of Computer Science
University at Buffalo
Buffalo, NY 14221
uneethak@buffalo.edu

Abstract

The task of this project is to build a reinforcement learning agent to navigate the classic 5x5 grid environment. I implemented reinforcement learning algorithm-Q-Learning. The agent will learn an optimal policy through Q-Learning which will allow it to take actions to reach a goal while avoiding obstacles. It finds the shortest path to reach the destination which is goal with maximum reward. The environment and agent is be built to be compatible with OpenAI Gym environments and will run effectively on computationally-limited machines.

1 Introduction

1.1 Reinforcement Learning

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize some notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

It differs from supervised learning in not needing labelled input/output pairs be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge). Reinforcement Learning lies between the spectrum of Supervised Learning and Unsupervised Learning, and there's a few important things to note:

Being greedy doesn't always work: There are things that are easy to do for instant gratification, and there's things that provide long term rewards. The goal is to not be greedy by looking for the quick immediate rewards, but instead to optimize for maximum rewards over the whole training.

Sequence matters in Reinforcement Learning: The reward agent does not just depend on the current state, but the entire history of states. Unlike supervised and unsupervised learning, time is important here.

The environment is typically stated in the form of a Markov decision process (MDP), because many reinforcement learning algorithms for this context utilize dynamic programming techniques. The main difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter do not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible.

An MDP is a 4-tuple (S, A, P, R), where

• S is the set of all possible states for the environment

• A is the set of all possible actions the agent can take

• P = Pr (s(t+1) = s' |s(t) = s, a(t) = a) is the state transition probability function

• R: S × A × S → R is the reward function

Our task is find a policy π: S → A which our agent will use to take actions in the environment which maximize cumulative reward, i.e.,

$$\sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1})$$

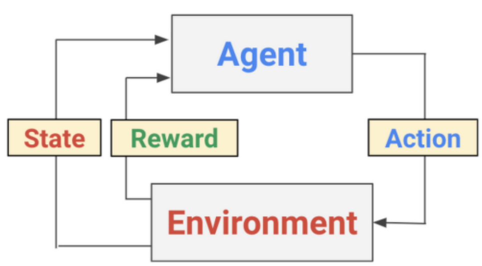
where γ ∈ [0, 1] is a discounting factor (used to give more weight to more immediate rewards), st is the state at time step t, at is the action the agent took at time step t, and st+1 is the state which the environment transitioned to after the agent took the action.

1.2 Q-Learning:

Q-learning is an off-policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward.

The 'q' in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward.

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



1.3 OpenAI Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball. The gym library is a collection of test problems — **Environments** — that you can use to work out your reinforcement learning algorithms. These environments have a shared interface, allowing you to write general algorithms. Here I am using one of the standard environment – GridEnvironment.

1.4 Hyper Parameters

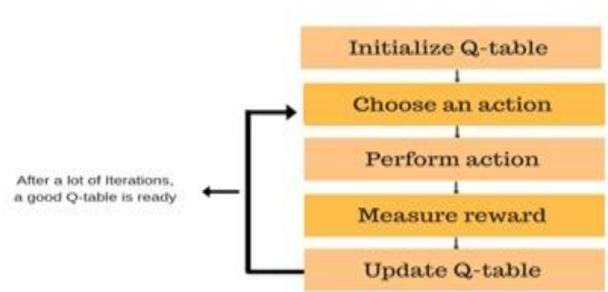
lr or learning rate, often referred to as *alpha* or α, can simply be defined as how much you accept the new value vs the old value. Above we are taking the difference between new and old and then multiplying that value by the learning rate. This value then gets added to our previous q-value which essentially moves it in the direction of our latest update.

Gamma also called as discount rate. It is mainly used to calculate the future discounted reward. The discount factor is mainly used to control the agent to how much area it should explore. If there is no discount rate the agent simply reaches the goal but takes lot of time. If there is a discount rate associated it reaches goal in quick and efficient manner by exploring all the ways. The value of discount factor should be less than 1.

Epsilon is the exploration/exploitation tradeoff. It is mainly used to control the amount the knowledge the agent gains. Initially we begin with larger value of epsilon by making the agent learn. Then we slowly decrease the value by changing to exploitation that is the agent has enough knowledge and now tries to maximize the reward.

Episode is one play that is agent moving from source to destination once. By increasing the number of episodes, the agent tries to learn more and gains high rewards. So, more episodes more the agent learn but at the same time with more episodes more time will be taken. Thereby improving performance of the agent and gaining the reward.

2 Learning System



In this task we use the concept of reinforcement learning. Reinforcement Learning (RL) is one of the machine Learning techniques that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences.

By following a certain sequence of steps, a learning system is selected for the problem. It allows the system to automatically learn and improve from experiences. The procedure or approach followed by this model is described by the steps as follows.

2.1 Agent

An agent takes actions. For eg, cab dropping off, drone making delivery, tom searching jerry. In our case Agent has to choose path with maximum reward to reach the Green box.

2.2 Q Table:

When q-learning is performed we create what's called a *q-table* or matrix that follows the shape of [state, action] and we initialize our values to zero. We then update and store our *q-values* after an episode. This q-table becomes a reference table for our agent to select the best action based on the q-value.

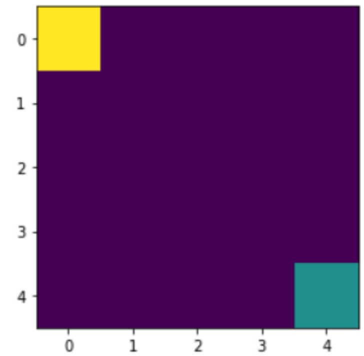
2.1 Grid Environment

The grid-world environment is first defined. This is the first step in implementation of the task. Environment here is a 5x5 grid, so a total of 25 steps are possible. Initially the environment is run just by using some random actions. This causes the agent to explore some actions thereby knowing some states, actions and gains rewards. There is no particular pattern followed here.

There are 3 important methods in the environment-

env.reset (): This will reset the environment and returns an initial observation.

env.render (): For every action taken in a particular step a window will be rendered.



env.step (): This returns four values that include reward, observation, a Boolean value and information regarding the action taken. By using all these methods we make the agent learn from the experiences and improve the performance by gaining high rewards.

> 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 done 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.

2.2 Making Updates

An agent interacts with the environment in 1 of 2 ways. The first is to use the q-table as a reference and view all possible actions for a given state. The agent then selects the action based on the max value of those actions. This is known as *exploiting* since we use the information we have available to us to make a decision.

The second way to take action is to act randomly. This is called *exploring*. Instead of selecting actions based on the max future reward we select an action at random. Acting randomly is important because it allows the agent to explore and discover new states that otherwise may not be selected during the exploitation process. You can balance exploration/exploitation using epsilon (ε) and setting the value of how often you want to explore vs exploit.

The Step function just calls the Policy Function.

2.3 Policy function:

$$\pi(s_t) = \operatorname*{argmax}_{a \in A} Q_\theta(s_t, a)$$

The policy function defines what step has to be taken. The epsilon is defined as 1.0 initially. The function generates a random number between 0 to 1.0. If the number is less than epsilon the agent uses the second method i.e., exploring. An action is selected at random. If the number is greater than epsilon, the agent uses the first method i.e., exploting. The agent selects the max value among the actions from the information available to us.

The epsilon value is multiplied with decay, 0.996 and compared with a min constant, 0.01 in my case. The maximum value is selected for each episode i.e. the epsilon is exponentially decayed and the reward is calculated. When the epsilon value is high, exploration is high. As the epsilon decays, the randomness decreases and the action with maximum reward is picked, i.e., exploitation is high.

2.4 Update f unction:

The q_table is updated with the formula given in section (1.2)

2.5 Training Process:

I initialized decay as 0.996 and episodes as 800. We are training the model with 800 episodes. For each episode the environment is reset and the actions are performed until the destination is reached. The epsilon value(decayed) is appended to the epsilons list for each episode and the reward is set to zero each time. Until the destination is not reached, the step function is called which in-turn calls the policy function, and the action to be performed is returned. The reward obtained is added to the overall reward calculated for the episode. Then I updated the q_table with the previous state, next_state and the reward obtained. Then the total reward calculated is appended to the rewards list. The Epsilon List and Rewards List is used for total rewards vs episode graph. The epsilon is calculated by multiplying with decay and min of the calculated value and 0.01 is chosen as epsilon (Exponential Decay.)

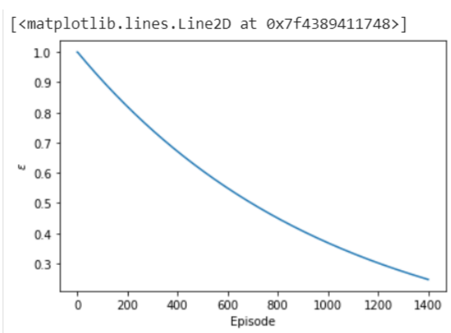
3 Results:

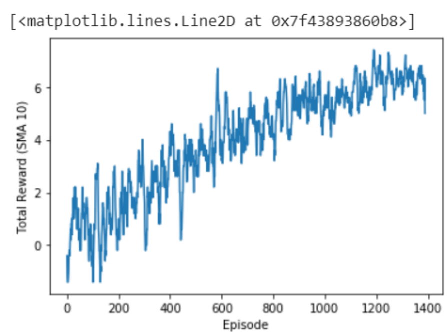
The Hyper parameters are learning rate (alpha), discount rate(gamma), epsilon and Episode. I trained the model with various hyper parameters and finally got optimum path with epsilon=1.0, lr=0.1, gamma=0.8, episodes=1400, decay=0.996.

With episodes =1400, decay=0.999 with other hyper parameters same as above

Did not converge

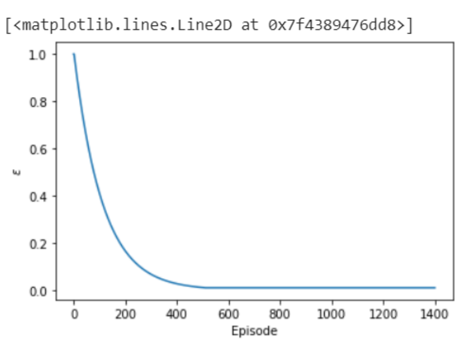
Epsilon Decay | Reward vs Episode

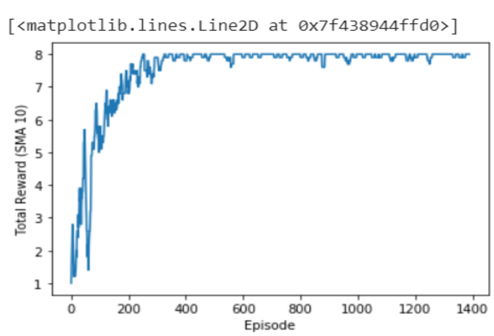




With episodes=1400, decay=0.991 Early Convergence

Epsilon Decay | Reward vs Episode

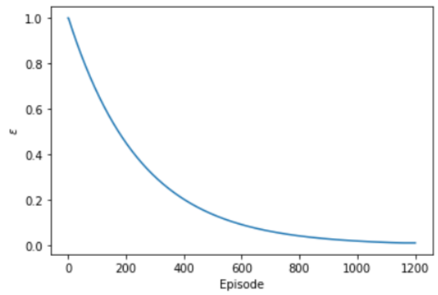




With episodes=1200 and decay=0.996 Optimum result is obtained.

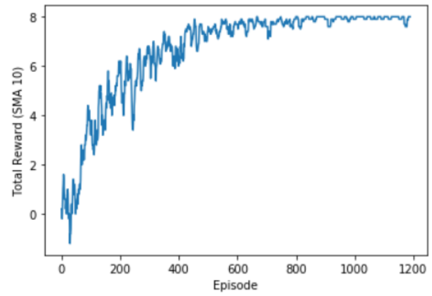
Epsilon Decay

[<matplotlib.lines.Line2D at 0x7f4388ab90f0>]



Reward vs Episode

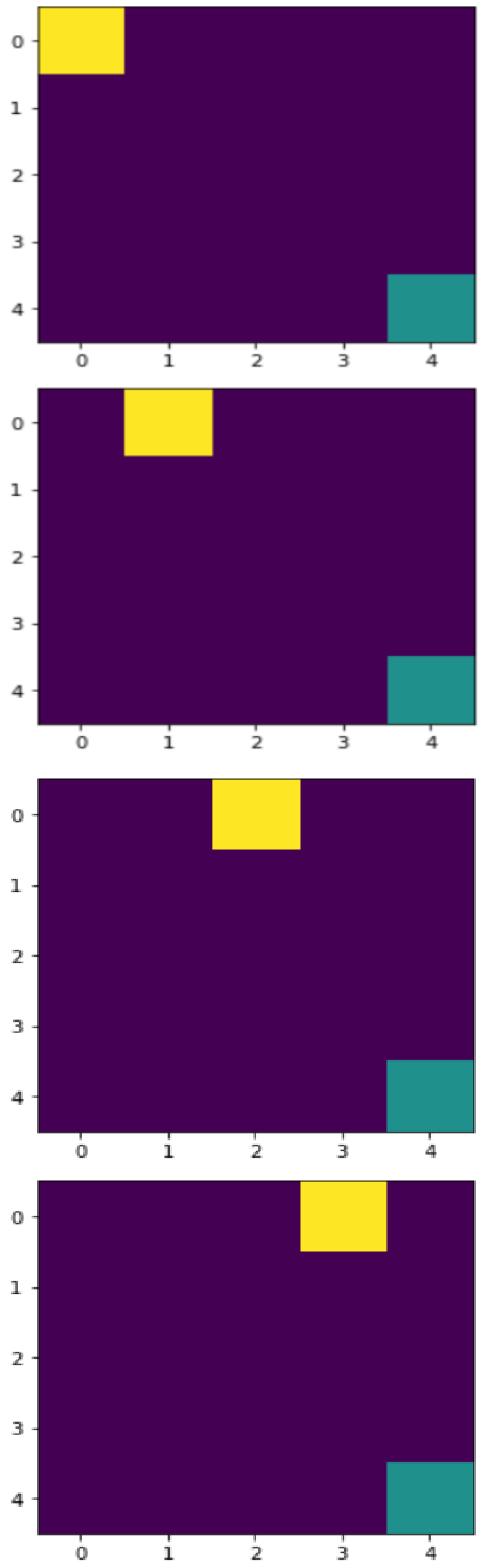
[<matplotlib.lines.Line2D at 0x7f43889ebeb8>]

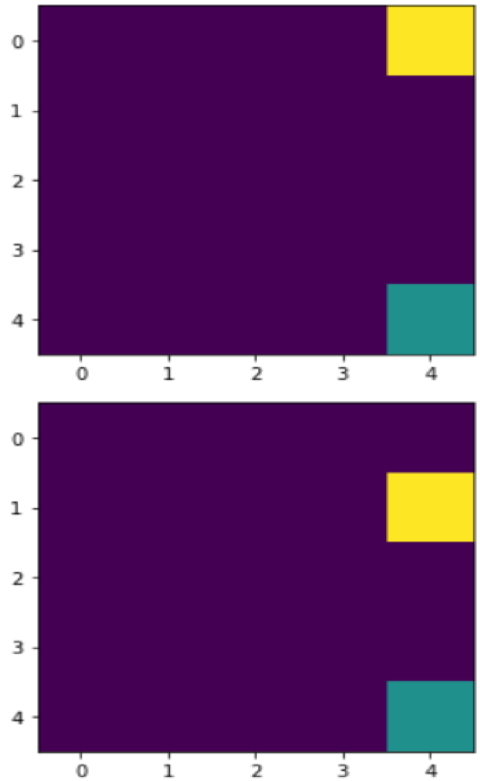


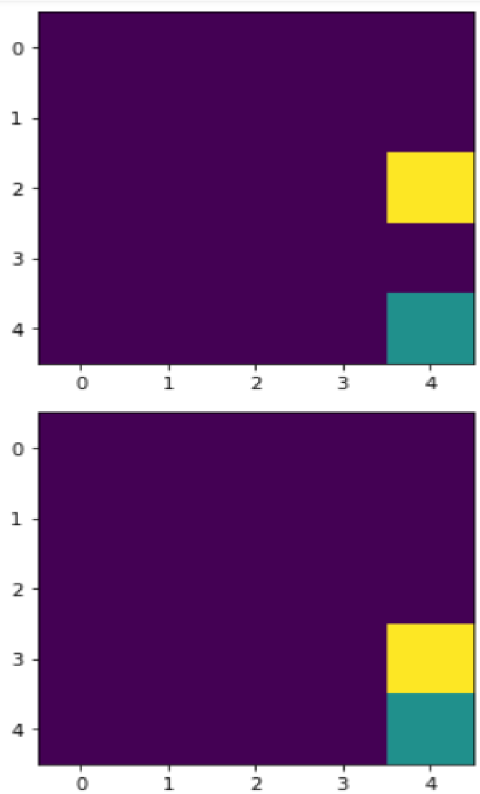
Q_table:

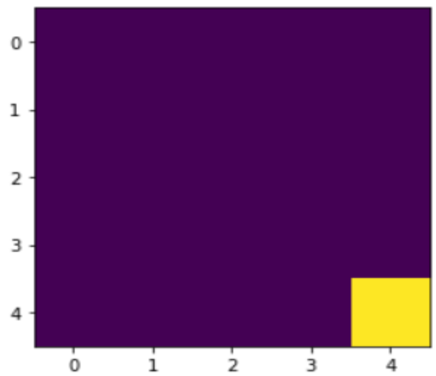
```
[[[ 5.6953279   3.96902407  5.35706371  3.97830536]
  [ 5.15015515  1.96460207  3.12193809  2.46434057]
  [ 4.5520628   0.51035565  1.01060649  0.46515947]
  [ 0.1        -0.25884837  1.05913464  0.41820297]
  [ 1.35580676 -0.40673677 -0.37418234 -0.18252754]]

 [[ 3.62422902  3.86461439  5.217031    3.46621215]
  [ 4.09798096  3.18028645  4.68559     3.4882741 ]
  [ 4.0951      2.74065283  2.83629613  3.11639693]
  [ 0.59901354 -0.18964892  2.88084433  0.39172489]
  [ 2.42684049 -0.38934159 -0.16693384 -0.0631441 ]]

 [[ 1.7607087   0.81888524  3.69795013  0.12476528]
  [ 3.90041607  1.44031874  2.15040194  0.14253487]
  [ 2.89528772  2.44343279  3.439       1.94102891]
  [ 2.71        1.02783178  2.09306921  1.83948222]
  [ 1.80947757 -0.022252    0.10274282  0.10698385]]

 [[ 1.6904757  -0.11004887  0.94913429 -0.36573544]
  [ 3.34997278  0.43203623  1.51095415 -0.01391113]
  [ 2.64156656  0.38319883  0.60970667  0.21217782]
  [ 1.9         1.12582835  1.57130081  0.94176705]
  [ 0.9774716  -0.08515917 -0.0631441  -0.07561   ]]

 [[-0.1112454  -0.074071    1.88423063 -0.29028758]
  [-0.17705596  0.30858445  2.68166717  0.01200704]
  [ 0.12286501  0.27184124  1.8992994  -0.02131716]
  [-0.1384689   0.41675677  1.          0.22066389]
  [ 0.          0.          0.          0.        ]]]
```

Optimal Path:




0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4


0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4



4 Conclusion:

Finally, the agent learns well and performs in a better way for the parameters tuned for the above environment and model. By changing the value of hyper parameters we compute the reward and the total time taken and then choose the values that give best reward.

5 References

[1] https://en.wikipedia.org/wiki/Reinforcement_learning Reinforcement Learning

[2]https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56 Q-Learning

[3] https://gym.openai.com/docs/ OpenAI Gym

[4] https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/ Reinforcement Q-Learning from Scratch in Python with OpenAI Gym

[5]https://www.google.com/search?q=reinforcement+learning&rlz=1C1CHBF_enUS864US864&sxsrf=ACYBGNQ3x6Y4bIJ_lUiJZ3Aw3_KaezF9-Q:1575513969203&source=lnms&tbm=isch&sa=X&ved=2ahUKEwi67-Pzvp3mAhWhrVkKHYXUB3EQ_AUoA3oECBIQBQ&biw=2304&bih=1010#imgrc=PKBmGSfjU0AvGM: Q-Learning Image