Project 4: Reinforcement Learning

Snehashish Mandal
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
snehashi@buffalo.edu

Abstract

The goal of this project is to build a reinforcement learning agent to navigate the classic 4x4 grid-world environment. The approach is to make the agent learn an optimal policy through Q-Learning which will enable the agent to take actions to reach the goal while avoiding obstacles. We are implementing the approach of tabular Q-Learning which uses a table of Q-values as the agent's policy. In brief, we are implementing the key components of the Q-learning algorithm. Specifically, we have been provided with a simple environment and a framework to facilitate training, and asked to provide the missing methods of the provided agent classes.

1 Introduction

Reinforcement learning is a machine learning paradigm which focuses on how automated agents can learn to take actions in response to the current state of an environment so as to maximize some reward. An autonomous agent must learn to perform a task by trial and error without any guidance from the human operator. This is typically modelled as a Markov Decision Process (MDP). Let's consider teaching a dog, we can not tell it what to do, but we can teach him to do the right thing by giving a reward for every right thing he does and a punishment for every wrong thing. The dog has to figure out own his own that what he did that made him get the reward or punishment, which is known as the credit assignment problem. [1][2]

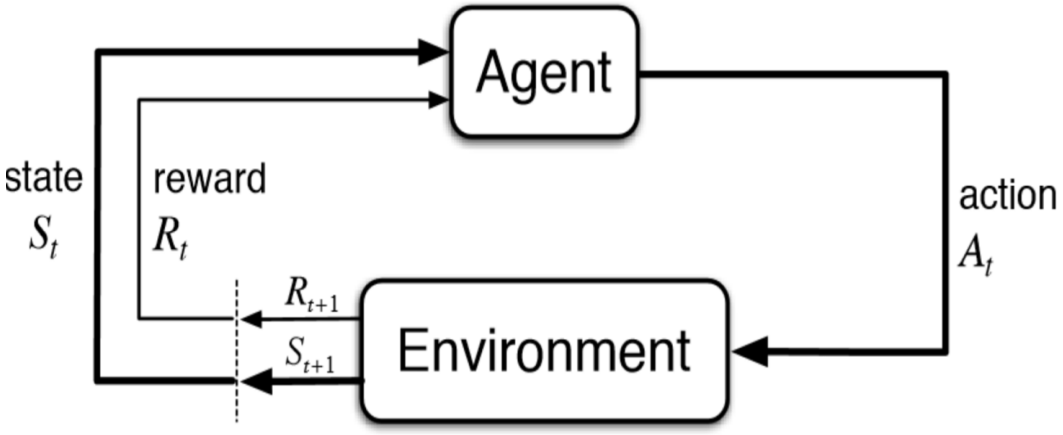

Figure 1. Canonical MDP diagram

$$S_0 \xrightarrow[R_0]{A_0} S_1 \xrightarrow[R_1]{A_1} S_2 \xrightarrow[R_2]{A_2} \cdots\cdots$$

An agent exists in an environment with set of states S, where it can perform any set of actions A, performing action A_t in state S_t receives a reward R_t. The agent's task is to learn a control policy, $\pi : S \rightarrow A$, that maximizes expected sum of rewards with future rewards discounted exponentially. Our goal is to learn to chose action that maximize the following:

$$R_0 + \gamma R_1 + \gamma^2 R_2 + ..., \quad \text{where } 0 \leq \gamma < 1$$

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

Where $\gamma \in [0; 1]$ is a discounting factor (used to give more weight to more immediate rewards), S_t is the state at time step t, at is the action the agent took at time step t, and S_{t+1} is the state which the environment transitioned to after the agent took the action. [2]

2 Implementation

2.1 Environment

Reinforcement learning environments can take on many different forms, including physical simulations, video games, stock market simulations, etc. The reinforcement learning community (and, specifically, OpenAI) has developed a standard of how such environments should be designed, and the library which facilitates this is OpenAI's Gym.

The environment provided to us is a basic deterministic n x n grid world environment where the agent has to reach the goal in least number of time-steps possible. Initially the agent is at (0,0) position and is going to work within an action space consisting of 4 actions: *up*, *down*, *left*, *right*. At each time step , the agent will take 1 step according to the action and move in the direction given by the action. For each action, the agent will receive a reward of either +1 or -1 depending on its position from the goal. +1 if the agent moves closer to the goal else -1. [1]

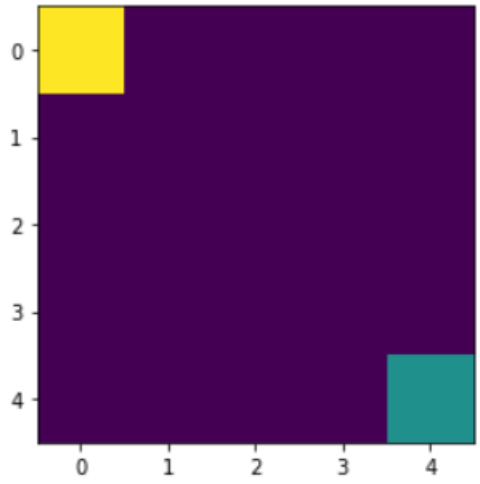

Figure 2. Initial setup of the environment (yellow: agent, green: goal)

First, we initialize our environment. The environment, loosely structured like OpenAI's Gym Environments, has three main methods: reset, step and render. When we call reset, we initialize the environment with a fresh episode. This allows us to effectively run through episodes (only needing to call reset at the beginning of an episode), but, more importantly, reset() returns the environment's initial state. The step method accepts an action as a parameter (which, for this example, is an integer in [0, 3]), processes the action, and returns the new state, the reward for performing the action, and a boolean indicating if the run is over.

2.2 Q-Learning

Q-Learning is a process in which we train some function Q_Θ : S X A -> R, parameterized by Θ, to learn a mapping from state-action pairs to their Q-value, which is the expected discounted reward for following the following policy Π_Θ:

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

In words, the function Q_Θ will tell us which action will lead to which expected cumulative discounted reward, and our policy Π will choose the action a which, ideally, will lead to the maximum such value given the current state S_t. Originally, Q-Learning was done in a tabular fashion. Here, we would create an |S| X |A| array, our Q-Table, which would have entries $q_{i,j}$ where i corresponds to the ith state (the row) and j corresponds to the jth action (the column), so that if S_t is located in the ith row and a_t is the jth column, $Q_{(St,\ at)} = q_{i,j}$. We use a value iteration update algorithm to update our Q-values as we explore the environment's states:

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

We are using our Q-function recursively to match (following our policy Π) in order to calculate the discounted cumulative total reward. We initialize the table with all 0 values, and as we explore the environment (e.g., take random actions), collect our trajectories, [s_0, a_0, r_0, s_1, a_1, r_2,…., s_T, a_T, r_T], and use these values to update our corresponding entries in the Q-table. [1]

2.3 Training

We initialized the environment and the agent for every episode. After initialization, we passed the initial state to obs by calling reset(). Then check if it's already done. If it is not done, we'll keep going. While it's not done, we'll need to update state, action,reward and next state. We can get action by step the current state on agent, used copy to record the current state, step the current action on environment to return the new state, the reward for performing the action, a Boolean done indicating if the run is over and some other information. Added the new reward on the total rewards. Used copy to save the new state returned by step. Updated the state, action, reward, next state of agent. This whole process is being done for number of episodes we have taken, i.e, 1000.

Apart from this, we are also updating the value of epsilon every episode. The epsilon is the exploration rate and initially it has been initialized with 1 and is being decayed every episode exponentially. When the epsilon value is greater, the agent chose some random action every step which allows it to explore actions that do not currently have high values. This process is called exploration and once the epsilon value is less than some value, the agent chose the action by looking at the q-table and returning the action which has maximum value. This process is called exploitation, where the agent exploits what it has learned and seek actions it believes will maximize its reward.

Rule for updating epsilon:

```
epsilon = 1
delta_epsilon = agent.epsilon/episodes
epsilon = agent.epsilon - (c * agent.epsilon * delta_epsilon)
    if epsilon < 0.1:
      epsilon = 0.1
```

Where c is a constant and it is varied to increase the decay rate of the epsilon. It has been varied during the process of tuning the hyper-parameters.

3.1 Tuning the hyper parameters

The hyper-parameters present here are: learning rate (α), discount factor (γ), exploration rate (ϵ) and the # of episodes. We have changed these values randomly and got different plots for epsilon vs episodes & rewards vs episodes and q-tables for each change. Those plots and q-tables have been added in results section.

4 Results

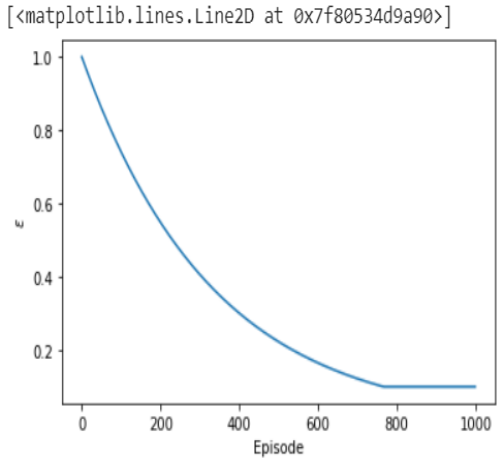

Figure 3. epsilon vs episodes (α = .1, γ = .9, c = 3)

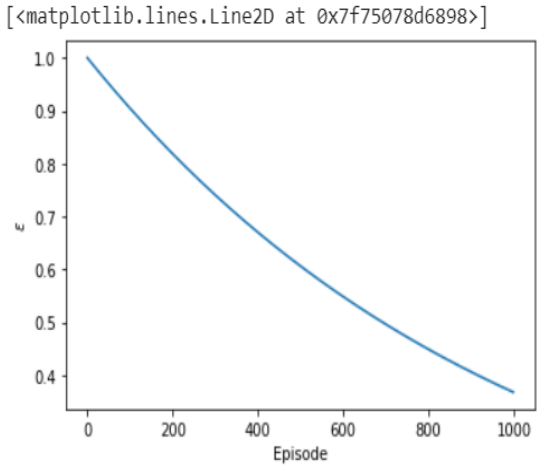

Figure 4. epsilon vs episodes (α = .1, γ = .9, c = 1)

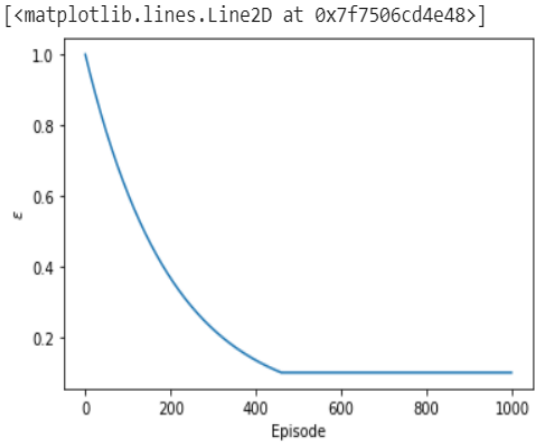

Figure 5. epsilon vs episodes (α = .1, γ = .9, c = 5)

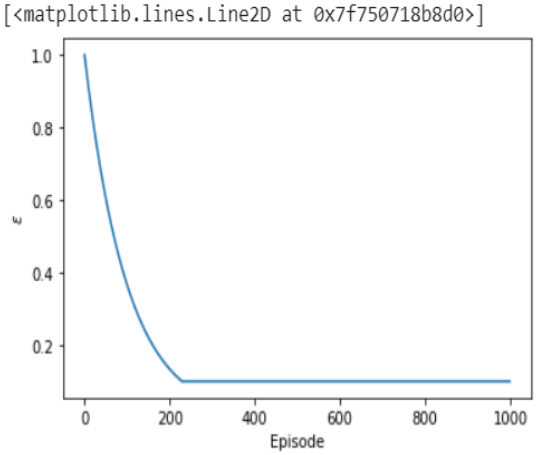

Figure 6. epsilon vs episodes (α = .1, γ = .9, c = 10)

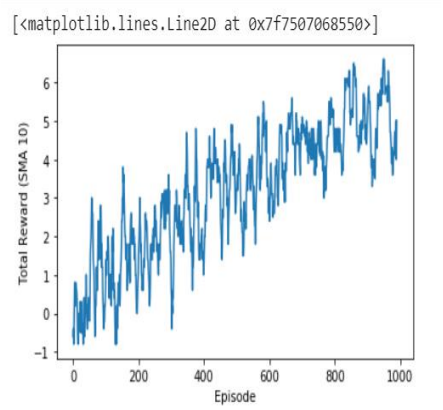

Figure 7. rewards vs episodes (α = .1, γ = .9, c = 1)

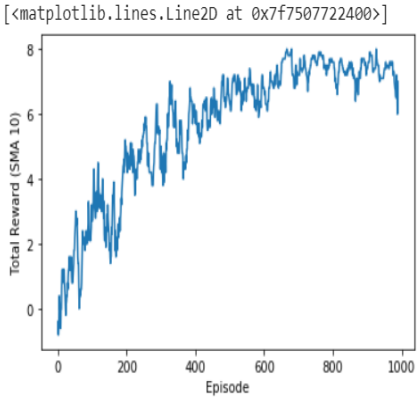

Figure 8. rewards vs episodes (α = .1, γ = .9, c = 3)

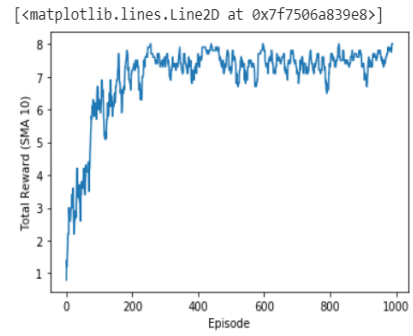

Figure 9. rewards vs episodes (α = .1, γ = .9, c = 10)

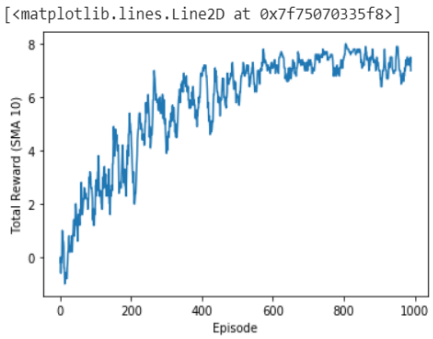

Figure 10. rewards vs episodes (α = .001, γ = .5, c = 3)

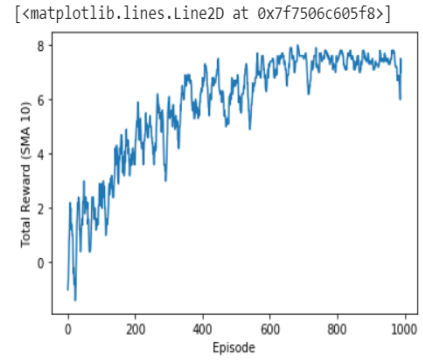

Figure 11. rewards vs episodes (α = .01, γ = 0, c = 3)

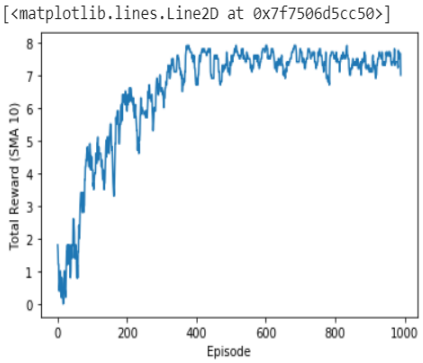

Figure 12. rewards vs episodes (α = .01, γ = 0, c = 5)

```
[ ]  agent.q_table
```

```
array([[[ 5.6953279 ,  4.05978074,  5.64600654,  4.10018387],
        [ 5.21388395,  2.4665175 ,  2.76212042,  3.29597321],
        [ 3.69903778, -0.0592234 ,  0.86939654,  0.37084281],
        [ 0.34989726, -0.40320487,  0.76335502,  0.01258579],
        [ 0.42661   , -0.436069  , -0.15022   , -0.05424949]],

       [[ 5.18482531,  4.02956776,  5.217031  ,  3.60326782],
        [ 4.68559   ,  3.58203718,  4.63130255,  3.61659817],
        [ 4.09327458,  0.82007688,  1.79633921,  1.88963976],
        [ 3.27411826, -0.35850758,  0.52097534,  0.70713522],
        [ 0.58906157,  0.        , -0.271     , -0.089461  ]],

       [[ 1.35339831,  1.95799289,  4.68435687,  2.04505986],
        [ 3.19261744,  3.16817998,  4.0951    ,  3.11344388],
        [ 3.32393591,  2.59788985,  3.439     ,  2.61609263],
        [ 2.71      ,  1.59071467,  1.8809724 ,  1.97884275],
        [ 1.79243919, -0.13159803, -0.19710329, -0.01011975]],

       [[ 0.62434208, -0.1833509 ,  1.83962427, -0.26867974],
        [ 2.75651042,  0.77629207,  1.64700189,  0.02231673],
        [ 0.33680301,  0.7142419 ,  2.69715312,  0.26909634],
        [ 1.80006975,  1.22775708,  1.9       ,  1.1360609 ],
        [ 1.        ,  0.33164508, -0.12022932,  0.63349797]],

       [[-0.0799759 , -0.06645222,  1.17312287, -0.06090956],
        [ 0.19242748,  0.05879603,  2.26025568, -0.20146063],
        [-0.20248133, -0.05022031,  1.73721153, -0.11491963],
        [-0.15333643, -0.01823036,  0.99363731, -0.01312321],
        [ 0.        ,  0.        ,  0.        ,  0.        ]]])
```

Figure 13. q_table (α = .1, γ = .9, c = 3)

```
array([[[ 5.69206300e+00,  4.10613709e+00,  5.66744951e+00,  4.10790803e+00],
        [ 5.20888834e+00,  3.37766734e+00,  4.82618536e+00,  3.84446553e+00],
        [ 4.64574150e+00,  2.29593596e+00,  2.14667740e+00,  2.70174953e+00],
        [ 1.27733069e+00, -1.45080449e-01,  1.78260398e+00,  1.12782908e+00],
        [ 1.45935461e+00, -2.81468500e-01, -1.67266845e-01,  -2.67167394e-02]],

       [[ 5.15618265e+00,  4.09558803e+00,  5.21430006e+00,  3.68024899e+00],
        [ 4.65953864e+00,  3.64908769e+00,  4.68338276e+00,  3.67541561e+00],
        [ 4.09323023e+00,  3.10334708e+00,  3.89602251e+00,  3.18126777e+00],
        [ 3.31227954e+00,  2.03185228e-01,  1.53215462e+00,  1.25779494e+00],
        [ 1.42375534e+00, -8.68832013e-02, -2.22772137e-01,  3.51020613e-01]],

       [[ 2.96685967e+00,  3.03042729e+00,  4.65983537e+00,  2.85582321e+00],
        [ 3.54331778e+00,  2.97871305e+00,  4.09019397e+00,  2.89543262e+00],
        [ 3.43749994e+00,  2.66476872e+00,  3.35798386e+00,  2.62686853e+00],
        [ 2.67820455e+00,  1.03843374e+00,  1.62835298e+00,  1.62776042e+00],
        [ 1.18078841e+00, -2.26488790e-01, -1.82409981e-01,  1.03821350e-01]],

       [[ 1.53723745e+00,  8.23030710e-01,  2.96276529e+00,  3.74724174e-01],
        [ 3.25388615e+00,  1.41670118e+00,  2.42108492e+00,  7.80853151e-01],
        [ 2.70882806e+00,  1.96371256e+00,  2.49118265e+00,  1.63881218e+00],
        [ 1.89168316e+00,  5.18541354e-01,  8.57838885e-01,  7.90219410e-01],
        [ 6.12579511e-01, -1.09012344e-01,  0.00000000e+00,  0.00000000e+00]],

       [[-1.59924309e-01,  6.56341608e-04,  1.98161573e+00,  2.28846999e-02],
        [ 2.34404340e-01,  8.72043263e-01,  2.63929651e+00,  2.44474135e-02],
        [ 6.27953087e-01,  1.35974606e+00,  1.89908542e+00,  1.06327831e+00],
        [-1.26158126e-01,  2.06141702e-01,  9.99303801e-01,  4.65901153e-01],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00]]])
```

Figure 14. q_table (α = .1, γ = .9, c = 1)

```
array([[[ 4.10484604e+00,  3.93071293e+00,  5.69532790e+00,  4.05674085e+00],
        [ 4.18862953e+00,  3.53727439e+00,  5.21703100e+00,  4.00282935e+00],
        [ 4.15656022e+00,  3.06566116e+00,  4.68559000e+00,  3.57862248e+00],
        [ 3.86734622e+00,  2.55915321e+00,  4.09510000e+00,  2.98017617e+00],
        [ 3.43900000e+00,  1.95995405e+00,  1.78861274e+00,  2.50948445e+00]],

       [[ 4.09465172e+00,  1.42231861e+00,  1.21731749e+00,  5.02241699e-01],
        [ 3.95875985e+00,  1.37406389e+00,  1.30041524e+00,  6.17615813e-01],
        [ 7.30579381e-01,  6.44568864e-01,  3.93208627e+00,  3.94250570e-01],
        [ 3.42513248e+00,  5.68355828e-01,  1.59924184e+00,  3.38970475e-01],
        [ 2.71000000e+00,  1.90670048e+00,  1.30157707e+00,  1.77283104e+00]],

       [[ 8.63634539e-01,  5.41771449e-01,  3.96266040e+00,  1.55631738e-01],
        [ 3.72610918e+00,  3.88607497e-01,  1.00918197e+00,  8.01973917e-01],
        [ 0.00000000e+00, -1.39612757e-01,  1.73746837e+00,  -1.49354282e-03],
        [ 2.70497324e+00,  4.86049313e-01,  5.03218333e-01,  -7.56272508e-02],
        [ 1.90000000e+00,  1.35108312e+00,  6.32428999e-01,  8.65979177e-01]],

       [[ 1.99000000e-01,  0.00000000e+00,  2.16971007e+00,  -2.05898697e-02],
        [ 3.23020354e+00,  2.70985717e-02,  2.88100000e-01,  2.04994933e-01],
        [ 6.68333016e-01, -5.35204900e-02,  0.00000000e+00,  -2.09861601e-02],
        [ 1.89839369e+00,  1.13259695e-01,  1.02486866e+00,  -2.58268691e-01],
        [ 1.00000000e+00,  6.06062729e-01, -8.57687856e-02,  5.27650355e-01]],

       [[-1.81000000e-01, -7.74688690e-02,  6.56814153e-01,  -1.65339574e-01],
        [-4.43989804e-02,  2.31039219e-01,  2.61779593e+00,  -3.95598224e-01],
        [ 9.57201624e-02, -2.28295900e-01,  1.87877423e+00,  2.72000407e-01],
        [-1.00000000e-01,  4.29134581e-02,  9.99588902e-01,  1.79815707e-02],
        [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00]]])
```

Figure 15. q_table (α = .1, γ = .9, c = 5)

```
array([[[ 5.6953279 ,  3.8384192 ,  4.67305989,  3.92411413],
        [ 4.67423814,  0.53713309,  1.18891433,  0.9431661 ],
        [ 2.32722955, -0.04926422,  0.60861072,  0.04668193],
        [ 1.63093425, -0.03597901,  0.19      , -0.1       ],
        [ 0.22834387,  0.        ,  0.        ,  0.        ]],

       [[ 5.217031  ,  3.85512284,  4.31594867,  3.52816357],
        [ 2.20857348,  0.32882144,  4.59993482,  0.21154055],
        [ 1.14862415, -0.05571467,  4.0690894 ,  0.89893936],
        [ 3.42801372, -0.25554483,  0.85195229,  0.6587158 ],
        [ 0.95064711, -0.24661   , -0.0713737 , -0.05818536]],

       [[ 3.6252225 ,  3.42191755,  4.68559   ,  2.99437787],
        [ 3.25947116,  2.66334438,  4.0951    ,  2.98307187],
        [ 3.439     ,  2.16214534,  3.00999197,  2.24560989],
        [ 1.21756813, -0.06533029,  2.70609807,  0.10630953],
        [ 1.89836575, -0.21590938, -0.04465393,  0.36061931]],

       [[ 0.78010256,  0.3217031 ,  3.64196827, -0.01003214],
        [ 3.41346527,  0.80608998,  0.53758   ,  0.28625754],
        [ 2.71      ,  1.83668374,  2.09507063,  1.60989403],
        [ 0.        ,  0.26578139,  1.83700368,  0.        ],
        [ 0.99937342, -0.07129373, -0.08663756,  0.0518412 ]],

       [[-0.10363137,  0.        ,  1.42762982,  0.        ],
        [ 0.14069493,  0.08264209,  2.70843369, -0.10452204],
        [ 0.56836206,  1.21592094,  1.9       ,  1.11689053],
        [-0.12052946,  0.26036893,  1.        ,  0.59446845],
        [ 0.        ,  0.        ,  0.        ,  0.        ]]])
```

Figure 16. q_table (α = .1, γ = .9, c = 10)

```
array([[[ 0.13042025, -0.11768592,  0.7572859 , -0.11903163],
        [ 0.08962553, -0.08110027,  0.70540303, -0.08679917],
        [ 0.07380561, -0.07360998,  0.64819257, -0.07538387],
        [ 0.59239217, -0.04536641,  0.05890525, -0.05852855],
        [ 0.04943472, -0.00890647, -0.01284887, -0.00559601]],

       [[ 0.027779  , -0.0232936 ,  0.10881418, -0.0321125 ],
        [ 0.12945926, -0.031216  ,  0.03776924, -0.0270787 ],
        [ 0.09722874, -0.02638353,  0.02408697, -0.0155656 ],
        [ 0.51504418, -0.04845222,  0.03691274, -0.05491854],
        [ 0.0639411 , -0.00689384, -0.01180558, -0.0087121 ]],

       [[ 0.00798003, -0.0078695 ,  0.02594121, -0.00793326],
        [ 0.01889535, -0.02332696,  0.11218203, -0.01479779],
        [ 0.17306518, -0.01467458,  0.02210421, -0.02328562],
        [ 0.45495418, -0.02639111,  0.04181998, -0.03205805],
        [ 0.07964001, -0.01084051, -0.00594965, -0.00856577]],

       [[ 0.00598602, -0.00398154,  0.0020005 , -0.002995  ],
        [ 0.01984494, -0.00296281,  0.00301392, -0.001996  ],
        [ 0.01988803, -0.01729258,  0.13050204, -0.00989812],
        [ 0.03168101, -0.02443842,  0.44739735, -0.02958732],
        [ 0.33048428, -0.0127311 , -0.0177192 , -0.02203594]],

       [[-0.001999  , -0.00299301,  0.00300299,  0.        ],
        [ 0.        , -0.000999  ,  0.00899345, -0.0019985 ],
        [-0.00298703, -0.00397507,  0.02089793, -0.00299301],
        [-0.00397513,  0.        ,  0.02470229, -0.00198654],
        [ 0.        ,  0.        ,  0.        ,  0.        ]]])
```

Figure 17. q_table (α = .001, γ = .5, c = 3)

```
array([[[ 1.        , -0.99999961,  0.99999989, -0.99999846],
        [ 0.99999992, -0.98922474,  0.94766524, -0.98802748],
        [ 0.56953279, -0.65132156,  0.94766524, -0.56953279],
        [ 0.92023356, -0.19      ,  0.40951   , -0.40951   ],
        [ 0.65132156, -0.3439    , -0.19      , -0.19      ]],

       [[ 0.99992382, -0.99996356,  1.        , -0.99999453],
        [ 1.        , -0.99949247,  0.99996721, -0.99978153],
        [ 0.83322818, -0.74581342,  0.9993038 , -0.81469798],
        [ 0.56953279, -0.65132156,  0.99949247, -0.5217031 ],
        [ 0.99363731, -0.71757046, -0.6861894 , -0.5217031 ]],

       [[ 0.99999166, -0.83322818,  0.90152291, -0.86491483],
        [ 1.        , -0.9962429 ,  0.99726107, -0.99661861],
        [ 0.99943608, -0.71757046,  0.6861894 , -0.61257951],
        [ 0.1       , -0.271     ,  0.71757046, -0.3439    ],
        [ 0.97972444, -0.3439    , -0.40951   , -0.5217031 ]],

       [[ 0.94766524, -0.83322818,  0.99999074, -0.79410887],
        [ 0.99778147, -0.99800332,  1.        , -0.99695675],
        [ 1.        , -0.98669721,  0.97972444, -0.99127204],
        [ 0.97972444, -0.40951   ,  0.40951   , -0.271     ],
        [ 0.81469798, -0.1       , -0.1       , -0.3439    ]],

       [[-0.56953279, -0.468559  ,  0.92023356, -0.61257951],
        [-0.74581342, -0.74581342,  0.99937342, -0.65132156],
        [-0.98521912, -0.89058101,  1.        , -0.9835768 ],
        [-0.79410887, -0.83322818,  1.        , -0.9282102 ],
        [ 0.        ,  0.        ,  0.        ,  0.        ]]])
```

Figure 18. q_table (α = .1, γ = 0, c = 3)

5 Conclusion

By changing the Hyperparameters, we checked for different parameters and finally set the parameters as Learning Rate = 0.1, Gamma = 0.9, Episodes = 1000, c = 3. Q-learning is an off-policy reinforcement learning algorithm. Q-learning uses future rewards to influence the current action given a state and therefore helps the agent select best actions that maximize total reward and reach the final state within a fair number of epochs. We are using our Q-function recursively to match (following our policy π) in order to calculate the discounted cumulative total reward.

References

[1] Project description PDF.

[2] Professor's slides