ECEN-743

REINFORCEMENT LEARNING

Assignment 3

Planning

1. **Q-Value Iteration (QVI)**: Implement Q-value iteration on the frozen lake environment.

   (a) What is the optimal policy and value function?

   Ans:

   Optimal Policy is given by the set { s: a} where s is state, a is optimal action for s

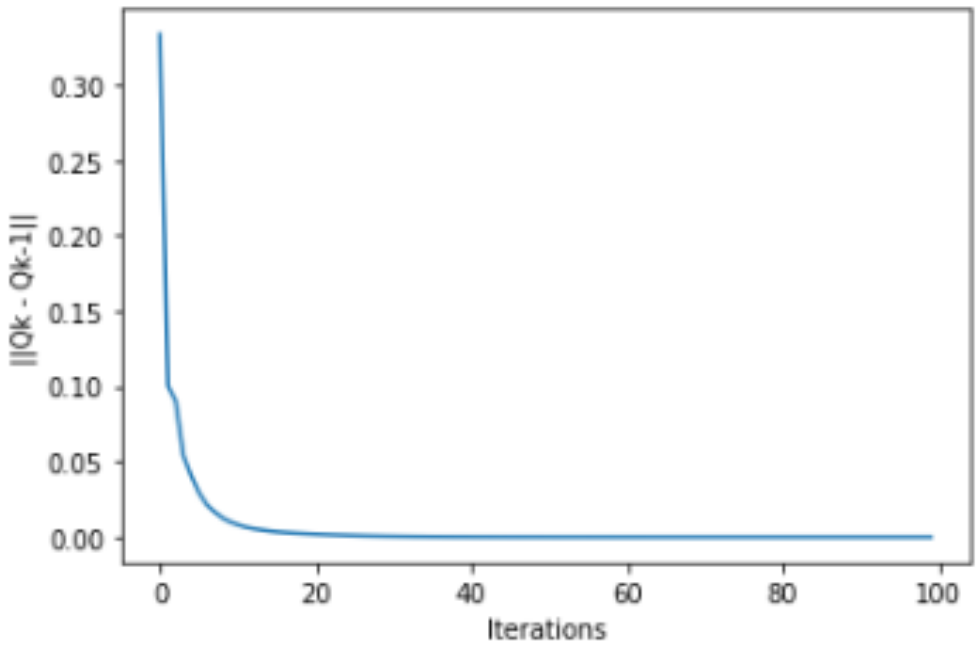
   ```
   {0: 0, 1: 3, 2: 0, 3: 3, 4: 0, 5: 0, 6: 0, 7: 0, 8: 3, 9: 1, 10: 0,
   11: 0, 12: 0, 13: 2, 14: 1, 15: 0}
   ```

   Optimal Value function is given by :

   ```
   [0.06889059, 0.06141431, 0.07440957, 0.05580711, 0.09185426, 0,
   0.1122081, 0, 0.14543612, 0.2474968, 0.29961748, 0, 0, 0.37993579,
   0.63902009, 0]
   ```

   (b) Plot Uk = ||Qk − Qk−1||, where Qk is the Q-value during the k th iteration.

   (Considering inf norm)



(c) Use the fancy visual function, which is available in the helper file to plot the heat maps of the optimal policy and value function.

---

2. **<u>Policy Evaluation</u>:** Consider the following polices: (i) the optimal policy obtained from QVI, and (ii) a uniformly random policy where each action is taken with equal probability.
   Compute the value of the these polices using:
   (a) By solving a linear systems of equations
      i. Using Optimal policy from QVI

```
[6.88909049e-02, 6.14145715e-02, 7.44097620e-02, 5.58073215e-02,
9.18545399e-02, 1.85037171e-16, 1.12208206e-01, 0.00000000e+00,
1.45436355e-01, 2.47496955e-01, 2.99617593e-01, 0.00000000e+00,
0.00000000e+00, 3.79935901e-01, 6.39020148e-01, 0.00000000e+00]
```

      ii. Using uniform random policy

```
[ 4.47726069e-03, 4.22245661e-03, 1.00667565e-02, 4.11821857e-03,
6.72195841e-03, 1.35138317e-16, 2.63337084e-02, 0.00000000e+00,
1.86761516e-02, 5.76070083e-02, 1.06971947e-01, 0.00000000e+00, -
5.69561934e-17, 1.30383049e-01, 3.91490160e-01, 0.00000000e+00]
```

(b) By the iterative approach

i. Optimal policy

```
[0.06889058, 0.0614143 , 0.07440956, 0.0558071 , 0.09185425, 0. ,
0.1122081 , 0. , 0.14543612, 0.2474968 , 0.29961748, 0. , 0. ,
0.37993579, 0.63902009, 0. ]
```

ii. Uniform Random Policy

```
[0.00447726, 0.00422246, 0.01006676, 0.00411822, 0.00672196, 0. ,
0.02633371, 0. , 0.01867615, 0.05760701, 0.10697195, 0. , 0. ,
0.13038305, 0.39149016, 0. ]
```

(c) Which method is better and why?

Ans: Iterative approach has time complexity of O(kS^2) where k is number of iterations till convergence and N is number of states and it is O(S^3) for solving systems of linear equations. In the case of frozen lake where S=16 and the number of iterations required to converge is found to be around 20, both are almost equal in terms of Big O notation (the latter is better if we are being accurate). However, in general when we expect S>>k, iterative method is better.

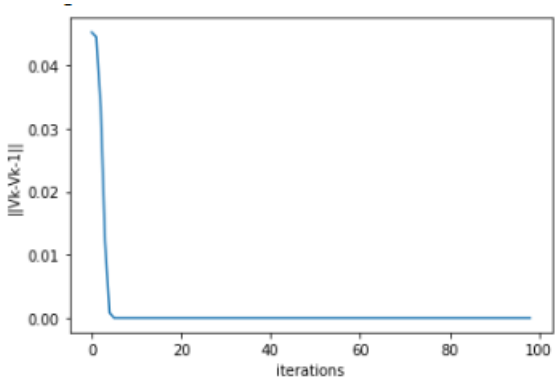
3. **Policy Iteration (PI):** Implement policy iteration on the frozen lake environment.
   a. What is the optimal policy and value function?
      Optimal Policy:

```
{0: 0, 1: 3, 2: 0, 3: 3, 4: 0, 5: 0, 6: 0, 7: 0, 8: 3, 9: 1,
10: 0, 11: 0, 12: 0, 13: 2, 14: 1, 15: 0}
```

      Value function:

```
[0.06889058, 0.0614143 , 0.07440956, 0.0558071 , 0.09185425,
0. , 0.1122081 , 0. , 0.14543612, 0.2474968 , 0.29961748, 0. ,
0. , 0.37993579, 0.63902009, 0. ])
```

   b. Compare the convergence of QVI and PI.

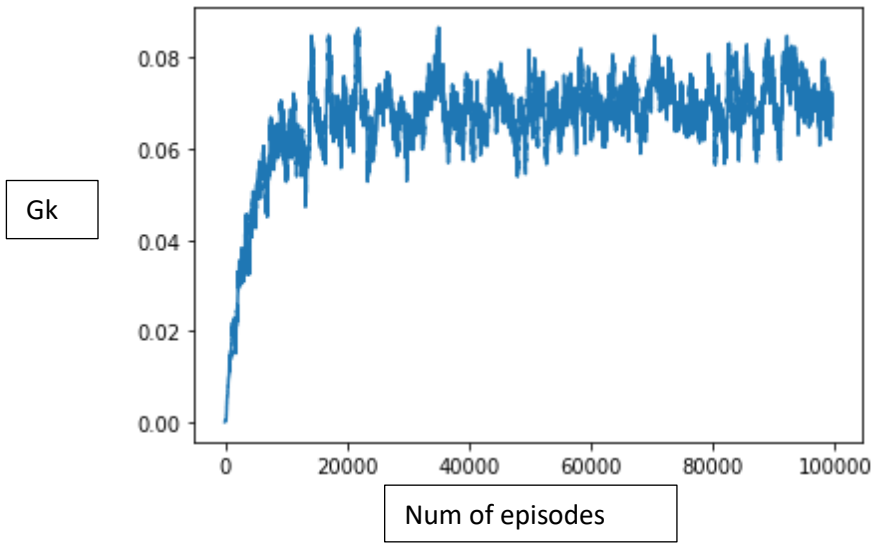


Both converge to same optimal policy. However, PI converges in fewer iterations (6 in this case) as compared to QVI(around 35).

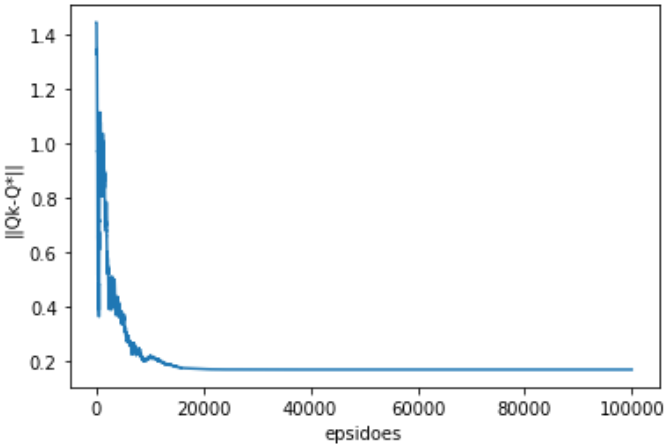
Learning

1. **Tabular Q-Learning**: Implement tabular Q-learning on the frozen lake environment. You should try to optimize the rate of convergence. Hint: Try various functions to decay exploration and learning rate. This will help you converge in lesser number of episodes.

   (a) Plot Gk, where Gk is the cumulative reward obtained in episode k. Use a sliding window averaging to obtain smooth plots.



   (b) Plot $||Qk - Q*||$, where k is the episode number and $Q*$ is the optimal Q function obtained from QVI.



   The plot is after smoothing with window size of 500.

(c) What is the policy and Q-value function obtained at the end of the learning? Are you able to learn the optimal policy?

Policy:

```
{0: 0, 1: 3, 2: 0, 3: 3, 4: 0, 5: 0, 6: 0, 7: 0, 8: 3, 9: 1, 10: 0,
11: 0, 12: 0, 13: 2, 14: 1, 15: 0}
```

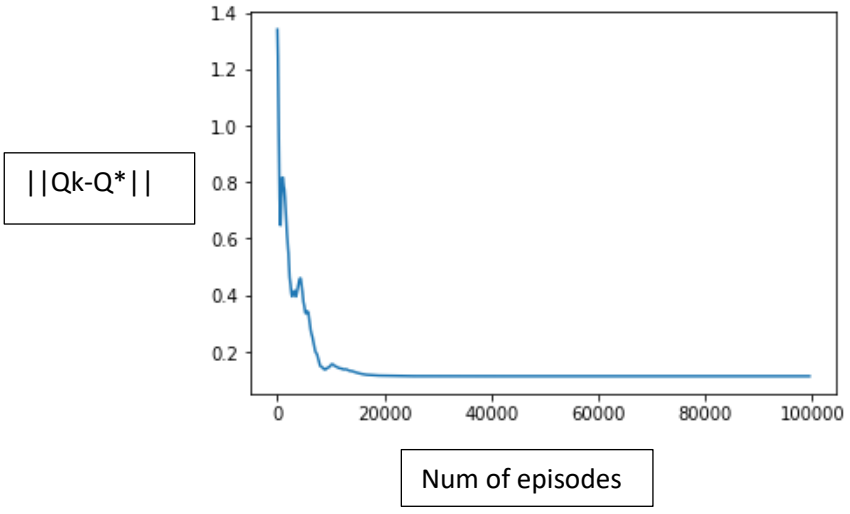
Q-value function:

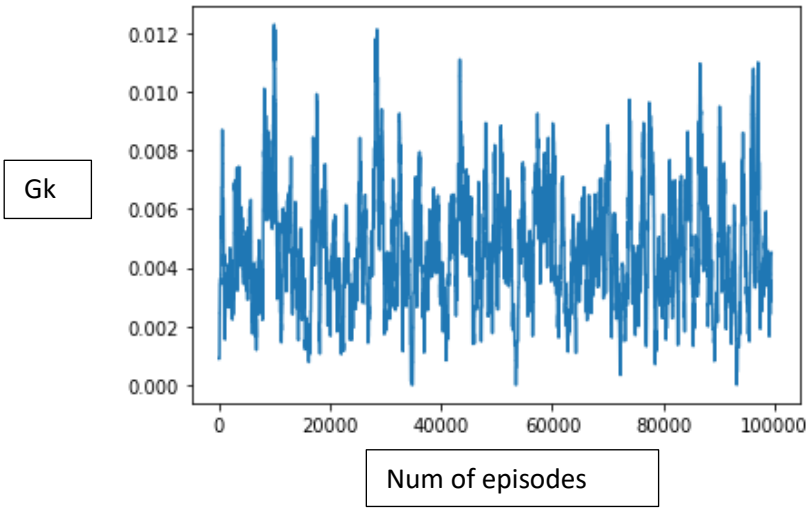
```
array([[0.06724288, 0.06508727, 0.06508727, 0.0581475 ],
       [0.03804647, 0.04207478, 0.03991917, 0.06002021],
       [0.07339314, 0.06777929, 0.0718076 , 0.05627906],
       [0.03833363, 0.03833363, 0.03271977, 0.05469352],
       [0.09035433, 0.0702533 , 0.06331353, 0.04714184],
       [0.        , 0.        , 0.        , 0.        ],
       [0.11165972, 0.08968598, 0.11165972, 0.02197374],
       [0.        , 0.        , 0.        , 0.        ],
       [0.0702533 , 0.11717851, 0.10100682, 0.14421931],
       [0.15698809, 0.24667407, 0.20346158, 0.13289847],
       [0.29903825, 0.26556428, 0.22507224, 0.10743998],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        ],
       [0.18774162, 0.30537387, 0.37933988, 0.26556428],
       [0.39505985, 0.6387072 , 0.61461758, 0.53679491],
       [0.        , 0.        , 0.        , 0.        ]])
```

Yes, we are able to learn the optimal policy via Tabular Q-learning.

2. Behavior policy: Implement tabular Q-learning with a uniformly random policy (where each action is taken with equal probability) as the behavior policy.
Compare the convergence with the ϵ-greedy exploration approach. Explain your observations and inference. Can you implement a better behavior policy and show its effectiveness?

Uniform Random Policy:





The plots are after smoothing with window size of 500.

It converges to a Q-value function:

```
array([[0.08275112, 0.08239116, 0.08239574, 0.07414707],
       [0.05033148, 0.05619552, 0.05685991, 0.08088915],
       [0.1020835 , 0.09483389, 0.10324428, 0.08166297],
       [0.05790232, 0.05752851, 0.04841419, 0.08170378],
       [0.11063177, 0.08751997, 0.0795587 , 0.05860786],
       [0.        , 0.        , 0.        , 0.        ],
       [0.14809223, 0.09388672, 0.14339371, 0.03166265],
       [0.        , 0.        , 0.        , 0.        ],
       [0.08687523, 0.139435  , 0.12283146, 0.17476901],
       [0.19295482, 0.28919807, 0.23976179, 0.16075982],
       [0.34308726, 0.28072353, 0.25902271, 0.13331267],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        ],
       [0.22224621, 0.31251456, 0.41728717, 0.29430167],
       [0.43735113, 0.64059068, 0.6316203 , 0.56884172],
       [0.        , 0.        , 0.        , 0.        ]])
```

Value function :

```
array([0.08275112, 0.08088915, 0.10324428, 0.08170378, 0.11063177,
       0.        , 0.14809223, 0.        , 0.17476901, 0.28919807,
       0.34308726, 0.        , 0.        , 0.41728717, 0.64059068,
       0.        ])
```

Policy:

```
{0: 0, 1: 3, 2: 2, 3: 3, 4: 0, 5: 0, 6: 0, 7: 0, 8: 3, 9: 1, 10: 0,
11: 0, 12: 0, 13: 2, 14: 1, 15: 0}
```

It doesn’t converge to the optimal policy as obtained from Q-Value Iteration as epsilon greedy policy does.

The convergence is slower compared to epsilon-greedy policy based method. This can be attributed to the fact that in a way, uniform random policy continues to explore even after there is some idea as to which action is better in a given state by picking random actions. On the other

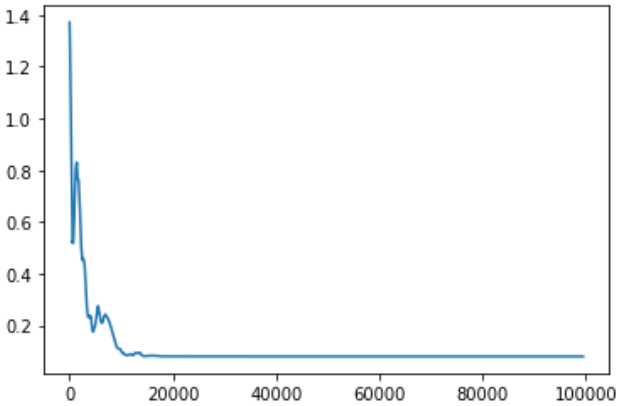
hand, ϵ-greedy exploration does exploration initially which then decreases over time and then sticks to the best actions. This makes it converge faster.

Better Policy:

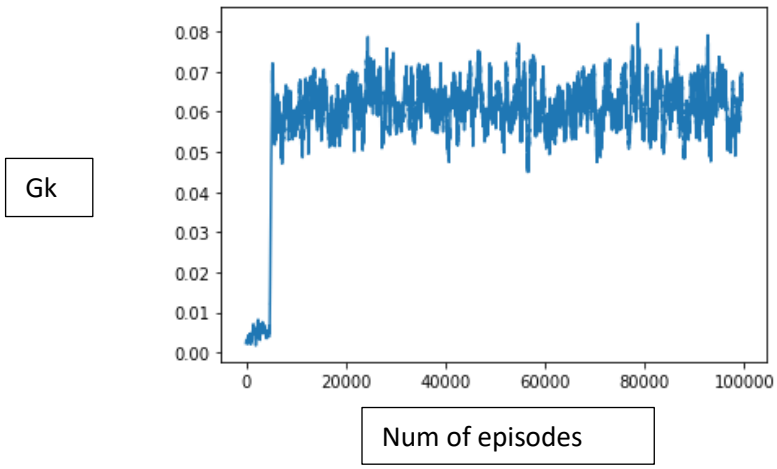
Greedy Policy: Choose action which always gives highest reward.
Q value func:

```
array([[0.04523622, 0.06223134, 0.04526117, 0.04520145],
       [0.0405725 , 0.03946076, 0.03915595, 0.05690543],
       [0.04410924, 0.04409575, 0.07018128, 0.04409249],
       [0.03328851, 0.03425061, 0.03389024, 0.05267851],
       [0.08849703, 0.0547195 , 0.05548899, 0.02470326],
       [0.        , 0.        , 0.        , 0.        ],
       [0.05550195, 0.05591351, 0.11134051, 0.03062688],
       [0.        , 0.        , 0.        , 0.        ],
       [0.07874035, 0.08213989, 0.0833208 , 0.14424681],
       [0.14984554, 0.2477773 , 0.14471471, 0.14367254],
       [0.29926267, 0.20778302, 0.16252012, 0.1221865 ],
       [0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        ],
       [0.24195288, 0.24213207, 0.38199689, 0.23358029],
       [0.41258866, 0.63677977, 0.45610501, 0.45731004],
       [0.        , 0.        , 0.        , 0.        ]])
```

Plot of ||Qk-Q*|| vs num of episodes:



Plot of Gk vs num of episodes:



The cumulative reward(around 0.07) is significantly higher than that of uniform random policy where it is around 0.012. In this way greedy policy is a better policy than uniform policy.

3. TD-Learning: Consider the following polices: (i) the optimal policy obtained from QVI, and (ii) a uniformly random policy where each action is taken with equal probability.

   Learn the value of the these polices using:

   (a) Monte Carlo (MC) Learning
      i. Optimal policy

   Value function:

   ```
   [0.06735581, 0.05621155, 0.06836778, 0. , 0.09024782, 0. ,
   0.1118974 , 0. , 0.14279557, 0.24530788, 0.29992916, 0. , 0. ,
   0.37567631, 0.6353193 , 0. ]
   ```

      ii. Uniform Random Policy

   Value function:

   ```
   [0.00411236, 0.00368242, 0.00734796, 0.00296496, 0.0061496 , 0. ,
   0.01784309, 0. , 0.01573661, 0.05007741, 0.08901134, 0. , 0. ,
   0.10830814, 0.3332299 , 0. ]
   ```

   (b) Temporal Difference (TD) Learning

      i. Optimal policy

   Value function:

   ```
   [0.06849869, 0.0615352 , 0.07446215, 0. , 0.09121501, 0. ,
   0.11273268, 0. , 0.14429797, 0.2451683 , 0.29715189, 0. , 0. ,
   0.37831932, 0.63729823, 0. ]
   ```

      ii. Uniform Random Policy

   Value function:

   ```
   [0.00435228, 0.00402701, 0.00954755, 0.0040572, 0.00659874, 0,
   0.02415587, 0, 0.01836029, 0.05684741, 0.09746036, 0, 0,
   0.13640199, 0.37019131, 0. ]
   ```

   (c) What are the trade-offs of between MC vs TD?

   MC method requires more experience but gives unbiased estimate and can converge to the true value, while TD methods require less experience but are biased and may not converge to the true values.