Reinforcement Learning

Sanidhya Chopde
Department of Computer Science
University at Buffalo
Buffalo, NY 14260
schopde@buffalo.edu

Abstract

The aim is to build a reinforcement learning agent to navigate the classic 4x4 world environment by learning an optimal policy through Q-learning.

1 Introduction

In this project we have to build a reinforcement learning agent which will navigate through the 4x4 environment, avoiding obstacles, to reach the goal. We will accomplish this task by training the agent in Q-learning policy which will allow the agent to take actions to reach the goal. Reinforcement learning is defined not by characterizing learning methods, but by characterizing a learning problem.

2 Reinforcement Learning

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.

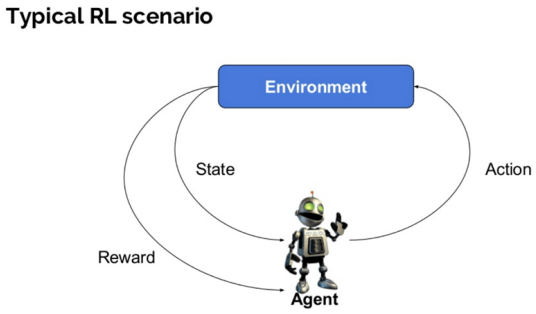


Figure 1: Typical scenario in Reinforcement learning.

3 Markov Decision Process

In a reinforcement learning problem, an agent is supposed to decide the best action to select based on his current state. When this step is repeated, the problem is known as a Markov Decision Process.

A Markov Decision Process (MDP) model contains:

- A set of possible world states S.
- A set of Models.
- A set of possible actions A.
- A real valued reward function R(s,a).
- A policy the solution of Markov Decision Process.

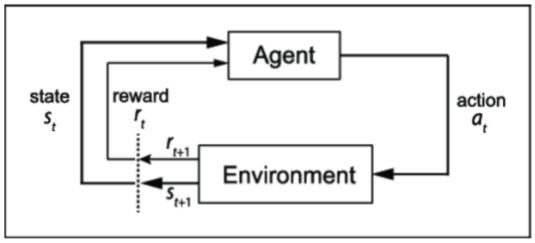


Figure 2: The canonical MDP diagram.

Here, the environment consists of states and we have an agent, performing actions, who gets rewarded positively or negatively when it reaches a particular state.

4 Environment

Reinforcement learning environments can take on many different forms, including physical simulations, video games, stock market simulations, etc. For this project we have chosen a basic deterministic n x n rid world environment.

The environment's state space will be described as an n × n matrix with real values on the interval [0, 1] to designate different features and their positions. The agent will 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.

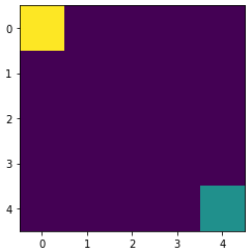


Figure 3: The initial state of our basic grid-world environment.

(the initial state for a 4 × 4 grid-world is shown in Figure 3. Here 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 reward for reaching closer to the final state is +1, the reward for reaching to a state which increases the distance between current position and the final state is -1 and the reward for every other action is -1.

5 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.

The Q-learning algorithm:

- Create a Q-table with the dimensions of [s, a] and initialize it with zero values.
- Now, the agent will perform actions in the environment and update the Q-table accordingly.
- Agents action can be either explore or exploit.
- The Q-table is updated after each action when the episode is done.

Following is the basic update rule for Q-learning:

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

6 Challenges

The major challenges faced are as follows:

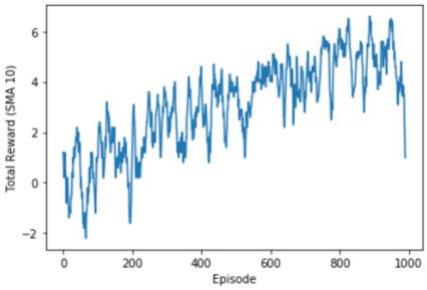
- Updating and maintaining the Q-table after every action.
- Calculating and generating results for each episode based on the updated rewards and epsilon values.
- The extra hyperparameter we show for episodes are: Epsilon value, Current Episode and the Previous reward. These hyperparameters get tuned over time and the performance is optimized.

7 Tabular Method Implementation

When the hyperparameters are:
episodes = 1000, lr = 0.1, epsilon = 1 and gamma = 0.9, we get the following Q-table:

```
[[[ 5.67313276  4.11689207  5.69306458  4.11672895]
  [ 5.21490203  3.68431922  5.19368098  4.10821361]
  [ 4.67786514  3.02571538  3.98039817  3.26590476]
  [ 4.03200544  0.91246816  2.76408291  1.84822378]
  [ 3.24653574  0.15493103  0.18524094  1.2119608 ]]

 [[ 4.2385075   3.9685241   5.21120431  3.55807802]
  [ 4.47325199  3.6830777   4.68366724  3.6732564 ]
  [ 4.02501336  3.16061718  4.09335595  3.20117795]
  [ 3.40109029  2.466805    3.43749147  2.6274434 ]
  [ 2.70889223  1.55203688  1.23940949  2.02176126]]

 [[ 1.67373493  2.44779046  4.16054063  1.60012963]
  [ 1.94807358  2.08937558  4.00727698  2.24005646]
  [ 1.77707261  2.0737107   3.41960621  2.0095391 ]
  [ 1.56177837  1.52219959  2.70561518  1.7860836 ]
  [ 1.89929769  1.17028089  0.61664249  1.2931333 ]]

 [[ 1.24135834  0.53544215  0.84459799 -0.3267996 ]
  [ 0.6261005   0.61907712  1.31202569 -0.11684807]
  [ 1.37395191  0.42844368  0.73044568 -0.16048473]
  [ 1.13076581  0.26277039  0.6656755  -0.1498526 ]
  [ 0.99963001  0.3801787  -0.14993828 -0.19915769]]

 [[-0.1709724  -0.12088101  0.68474206 -0.23248515]
  [-0.1379647  -0.05795441  0.81047043 -0.17209   ]
  [-0.23041    -0.162532    1.09956331 -0.19813954]
  [-0.21487858 -0.08634431  0.6861894   0.        ]
  [ 0.          0.          0.          0.        ]]]
```

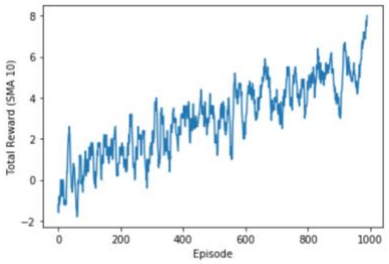
The graph for Episode vs Total reward is as follows:



When the hyperparameters are:
episodes = 1000, lr = 0.0001, epsilon = 1 and gamma = 0.85, we get the following Q-table:

```
[[[ 0.03316496 -0.03222723  0.09684104 -0.0335481 ]
  [ 0.07087497 -0.02170853  0.02048579 -0.02238473]
  [ 0.01389194 -0.00417607  0.00539319 -0.00607032]
  [ 0.0044044  -0.0021954   0.00139954 -0.00149372]
  [ 0.00090039 -0.00029995 -0.00059972 -0.00029975]]

 [[ 0.02819371 -0.0120436   0.01123696 -0.01113283]
  [ 0.04966606 -0.01492024  0.01219312 -0.01460085]
  [ 0.00502094 -0.005948    0.01623364 -0.00394121]
  [ 0.01060669 -0.00348697  0.00419606 -0.00287602]
  [ 0.00360212 -0.00029991 -0.00109864 -0.00149487]]

 [[ 0.01970979 -0.00759467  0.00862533 -0.00920054]
  [ 0.00829729 -0.00978074  0.03503355 -0.00770582]
  [ 0.00550078 -0.00584771  0.02515339 -0.00561564]
  [ 0.00439865 -0.00357861  0.01790581 -0.00465436]
  [ 0.00986833 -0.00269205 -0.00288807 -0.00238226]]

 [[ 0.00389607 -0.00287761  0.01376406 -0.00575785]
  [ 0.00209935 -0.00315808  0.01186642 -0.00437217]
  [ 0.00838402 -0.00168929  0.00210117 -0.00278284]
  [ 0.00399649 -0.00089653  0.00070116 -0.00059864]
  [ 0.00488826 -0.00049881 -0.00059909 -0.00129656]]

 [[-0.0003996  -0.00109449  0.00259842 -0.00129796]
  [-0.0004997  -0.00059797  0.0023019  -0.00089837]
  [-0.00169558 -0.00109484  0.00519411 -0.00169709]
  [-0.00079905 -0.00069922  0.00329473 -0.00049908]
  [ 0.          0.          0.          0.        ]]]
```

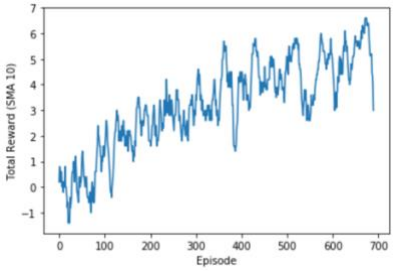
The graph for Episode vs Total reward is as follows:



When the hyperparameters are:
episodes = 700, lr = 0.005, epsilon = 1 and gamma = 0.95, we get the following Q-table:

```
[[[ 0.90425301 -0.04164215  2.18564874 -0.02328135]
  [ 0.74212874 -0.08849463  1.78689864  0.0192669 ]
  [ 1.40591176 -0.14924761  0.53529061 -0.04815803]
  [ 0.50598561 -0.17080112  0.11215663 -0.07746032]
  [ 0.09250363 -0.0286928  -0.04722126 -0.02645298]]

 [[ 0.32015962 -0.10701183  0.79555755 -0.16327901]
  [ 0.82821338 -0.08032104  0.32804023 -0.1874901 ]
  [ 0.31352353 -0.0826423   1.06969592 -0.14779437]
  [ 0.81943147 -0.19844346  0.23585351 -0.12723953]
  [ 0.1958783  -0.05599888 -0.03228751 -0.00928125]]

 [[ 0.26981354 -0.10381452  0.07918717 -0.08812627]
  [ 0.55247148 -0.10513849  0.15006086 -0.1778303 ]
  [ 0.09006853 -0.05577875  0.27516792 -0.0753714 ]
  [ 0.54754071 -0.09678988  0.17496717 -0.11799091]
  [ 0.18660583 -0.03111539 -0.05877465 -0.03612048]]

 [[ 0.14134149 -0.0414569   0.09705586 -0.06412394]
  [ 0.36733329 -0.09271644  0.09996388 -0.07110544]
  [ 0.0105861  -0.01856927  0.12538832 -0.01343045]
  [ 0.31896857 -0.06340264  0.10930667 -0.067978  ]
  [ 0.11334649 -0.02268897 -0.03257562 -0.02548805]]

 [[-0.03706545 -0.02393141  0.12297458 -0.01862344]
  [-0.09207444 -0.07565993  0.23614654 -0.05511901]
  [-0.05097667 -0.03210613  0.13880257 -0.01372433]
  [-0.02812751 -0.02086726  0.15669486 -0.02334898]
  [ 0.          0.          0.          0.        ]]]
```

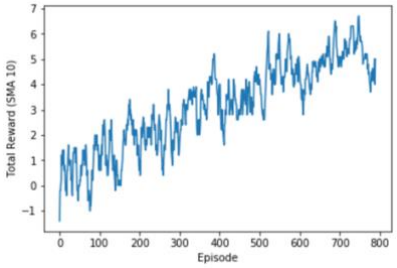
The graph for Episode vs Total reward is as follows:



When the hyperparameters are:
episodes = 800, lr = 0.77, epsilon = 1 and gamma = 0.9, we get the following Q-table:

```
[[[ 5.6953279   4.12579511  5.6953279   4.12579511]
  [ 5.217031    3.6953279   5.21703089  4.12579511]
  [ 4.68559     3.2170296   4.68558564  3.69531157]
  [ 4.09509902  2.67639744  4.09508367  3.21295909]
  [ 3.43899999  2.07651091  2.07156959  2.68362054]]

 [[ 5.217031    4.12579511  5.217031    3.6953279 ]
  [ 4.68559     3.6953279   4.68559     3.6953279 ]
  [ 4.0951      3.21703096  4.0951      3.21703083]
  [ 3.43899993  2.68558577  3.439       2.68559   ]
  [ 2.71        2.09509964  1.43899846  2.09509962]]

 [[ 4.68559     3.6953279   4.68559     3.217031  ]
  [ 4.0951      3.217031    4.0951      3.217031  ]
  [ 3.439       2.68559     3.439       2.68559   ]
  [ 2.71        2.0951      2.71        2.0951    ]
  [ 1.9         1.43899997  0.71        1.439     ]]

 [[ 3.06203739  3.21426882  4.0951      2.68473326]
  [ 3.37932681  2.68559     3.439       2.68555594]
  [ 2.69842511  2.09509996  2.71        2.09507386]
  [ 1.88847738  1.43876548  1.9         1.43899998]
  [ 1.          0.70999957 -0.1        0.70998496]]

 [[-0.12407849  2.63852478  0.987833    0.03691677]
  [ 1.27666428  2.06960806  2.64983678  0.03691677]
  [ 0.65378522  1.3715256   1.88847568  0.88999804]
  [-0.3314927   0.          0.987833   -0.14869019]
  [ 0.          0.          0.          0.        ]]]
```

The graph for Episode vs Total reward is as follows:



8 Results

In this project we trained our agent for four iterations and tuned the hyperparameters to get an optimized system. We can see from the above graphs and Q-table how on changing the hyperparameters the system got changed in every iteration. We found that with the ideal hyperparameters, the agent performs better and gives better results.

9 Conclusion

In this project we can conclude that q-learning maximizes total reward by influencing current actions based on future rewards. We saw how using different values for learning rate, gamma and no of episodes gave us different results. Increase the learning rate makes the learn faster but higher learning rates can lead to less exploration. We also use gamma to penalize rewards in the future which is very helpful in showing the importance of current state. As we go on tuning the hyperparameters, we reach a point when they seem ideal and our system gives us better results than it gave in the previous iterations and doing more iterations doesn't give good results. At this point we can stop iterating as we have found an optimal state for our system.

Acknowledgements

I am extremely grateful to Professor Sargur Srihari for teaching all the necessary concepts related to reinforcement learning and Q-learning. I would also like to thank all the TA's of the course for helping me at every step during the course of this project.

References

[1] Professor Sargur Srihari's lecture slides.
[2] Markov decision process (https://www.geeksforgeeks.org/markov-decision-process/)
[3] Q-learning (https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56)
[4] Reinforcement Learning(https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690)
[5] Q-learning(https://stats.stackexchange.com/questions/326788/when-to-choose-sarsa-vs-q-learning)
[6] Reinforcement Learning(https://www.analyticsvidhya.com/blog/2018/09/reinforcement-learning-model-based-planning-dynamic-programming/)