Training agent using Q-Learning in Grid World

Prudhveer Reddy Kankar
50320121
prudhvee@buffalo.edu
Department of Computer Science
University at buffalo

Abstract

In this project, we will build a reinforcement learning agent that will navigate the 4x4 grid environment. The agent will continuously learn from Q learning policy which will eventually allow the agent to take correct steps to reach the goal by avoiding obstacles. We will implement tabular Q-Learning, which is an approach that will utilize a table of Q-values as the agent's policy.

1. Introduction

1.1 Reinforcement Learning

Reinforcement learning is an area of Machine Learning. Reinforcement. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of training dataset, it is bound to learn from its experience.

Reinforcement learning can be understood using the concepts of agents, environments, states, actions and rewards.

Agent: Agent basically takes all the actions; thus, the our algorithm is the agent.

Action: If A is considered to be the set of all possible moves the agent can make, a move will be chosen from this list of discrete, possible actions.

Environment: The world through which the agent moves, and which responds to the agent. The environment takes the agent's current state ad action as input and returns as output the agent's reward and it's next state.

State: A state is a concrete and immediate situation in which the agent finds itself.

Reward: A reward is the feedback by which we measure the success or failure of an agent's action in a given state. These rewards, if any, will effectively evaluate the agent's action.

Our task is to find a policy, which the agent can use to take actions in the environment which maximizes the cumulative reward.

1.2 Markov decision process (MDP)

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

- S is the set of all possible states for the environment.
- A is the set of all possible actions the agent can take.
- P = P r(st+1 = s 0 |st = s, at = 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.

1.3 Q-learning algorithm

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.

Q-Learning is basically an for assessing the quality of the action which have to be taken to move to a state rather than determining the possible value of the state being moved to. R(s,a) is a metric which allots rewards for moving to a certain state. The Q values are the qualities of actions. Originally, Q-Learning was done in a tabular fashion. Here, we would create an | S| × |A| array, our Q-Table, which would have entries qi,j where i corresponds to the ith state (the row) and j corresponds to the jth action (the column), so that if st is located in the ith row and at is the jth column, Q(st, at) = qi,j . We use a value iteration update algorithm to update our Q-values as we explore the environment's states:

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

1.4 Q-Learning Agent

We perform the Q-Learning algorithm using this agent. The agent starts in a state, then takes an action and then receives a reward depending on the action taken. Agent selects action by referencing Q-table with the max value or by epsilon, ε. Then it updates the q-values.

2. Environment

2.1 Step

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 Reset

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.

2.3 Render

Render is used to display our environment.

3. Policy

The Q-Learning policy is implemented in the following way:

a. We will first select a random uniform number between 0 and 1 and if the number is less than the epsilon, then we choose a random action from the action space.

b. If the number is not less than epsilon, we choose the action based on the maximum reward calculated from the q learning table. This can be achieved by using the np.argmax, which returns the indices that contain the maximum values.

4. Update

Update function is used to update the q learning table based on the action that the agent takes.

5. Training

In training we initially reset the environment so as the agent is in the initial state.

From this we select the next action of the agent based on its policy function and after obtaining the next state that the agent should take, we pass this to the environment step function which executes the step and returns the next state and the reward obtained by the agent.

Using the next state, the q table of the agent is updated and this process is continued for 1000 episodes.

6. Hyperparameters

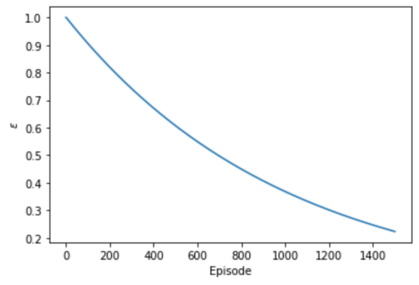
The hyperparameters of our program are

- Learning rate
- Gamma
- Episodes
- Decay Rate

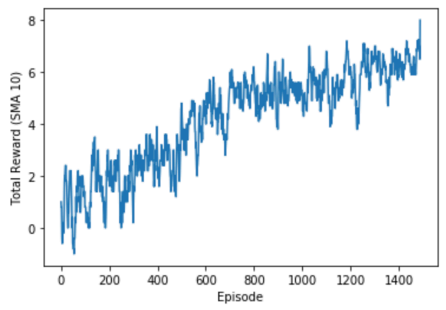
Changing this hyper parameters I was able to improve the performance of the model and these are some of the graphs that I obtained while tuning the hyper parameters.

For various hyper parameters values I obtained the following graphs while trying to get the best graph:

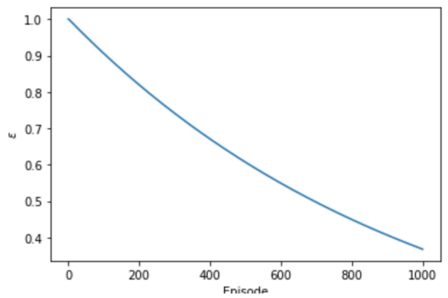
```
[<matplotlib.lines.Line2D at 0x7ff58cd8c198>]
```



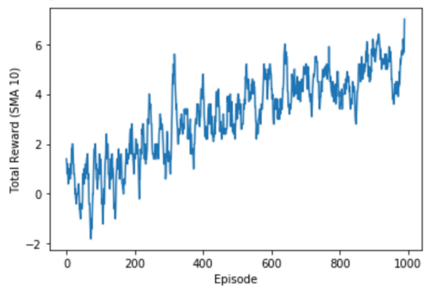
```
[<matplotlib.lines.Line2D at 0x7ff58d0b1400>]
```



```
[<matplotlib.lines.Line2D at 0x7ff58cddb780>]
```



```
[<matplotlib.lines.Line2D at 0x7ff58d45e7f0>]
```

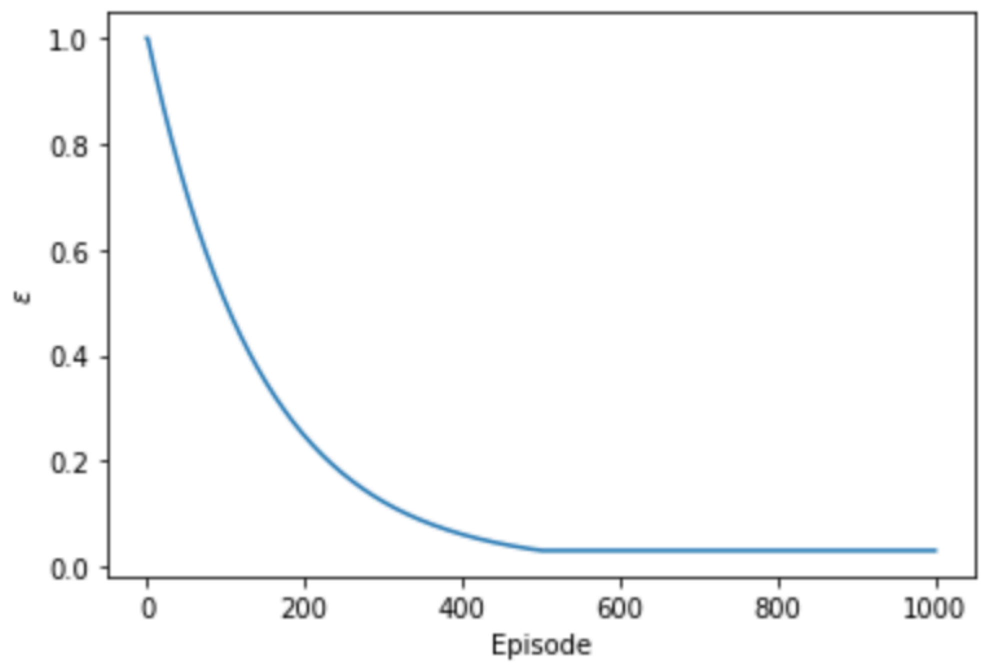


7. Results

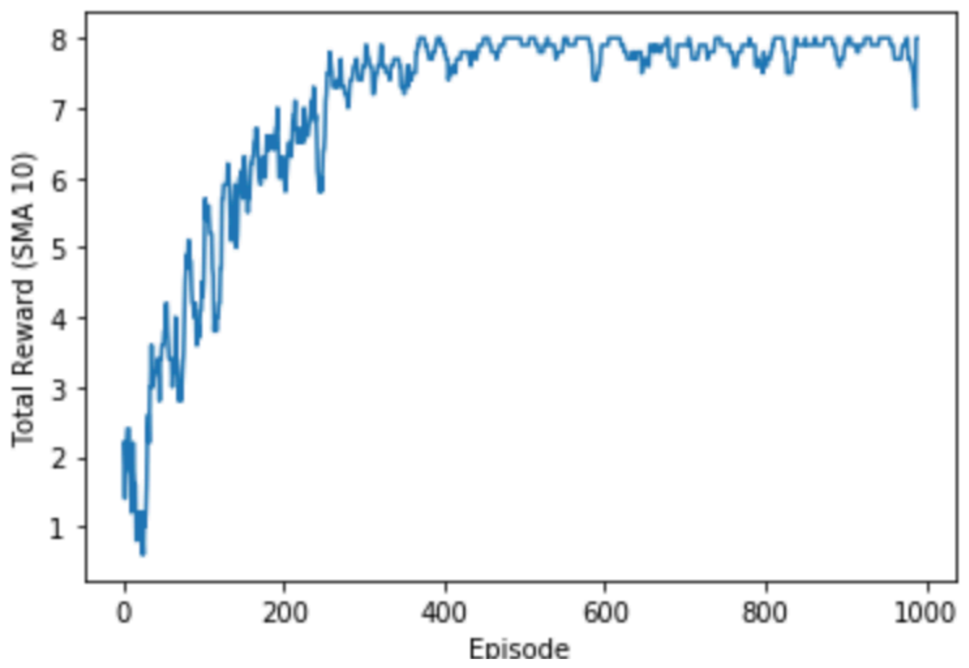
These are the best possible graphs I can get from this model for the following values:

- Learning rate - 0.1
- Gamma - 0.9
- Episodes - 1000
- Decay Rate - 0.997

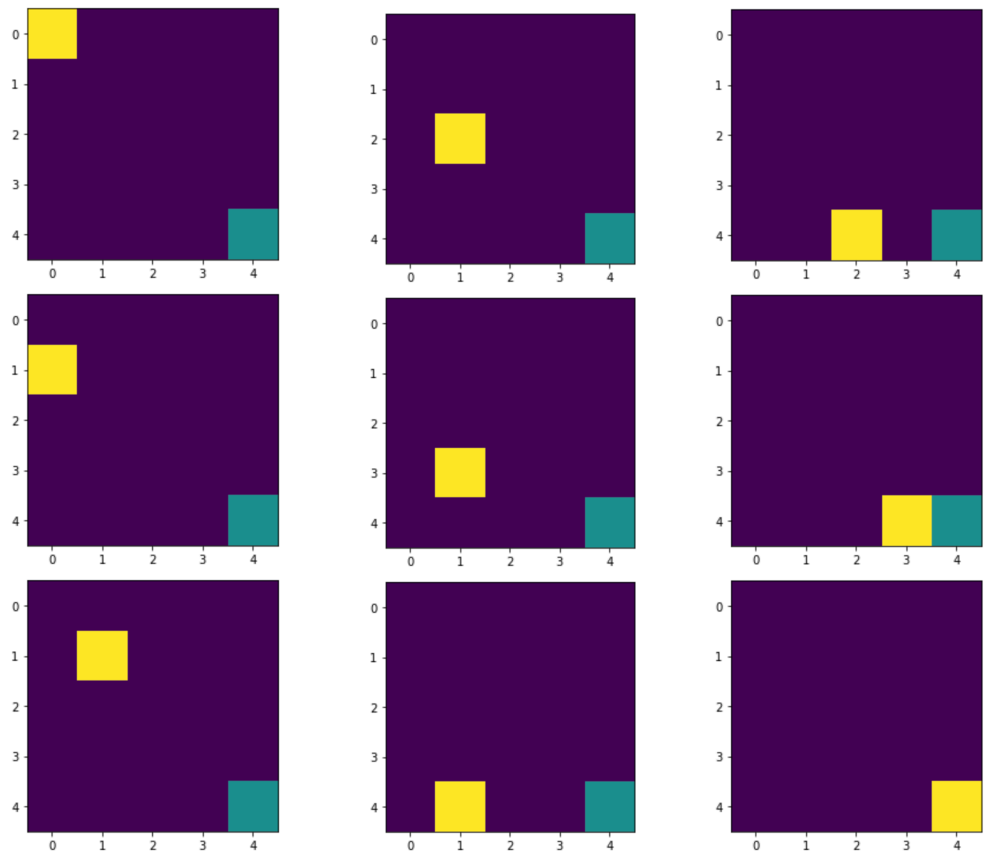
```
[<matplotlib.lines.Line2D at 0x7f2ca5a602e8>]
```



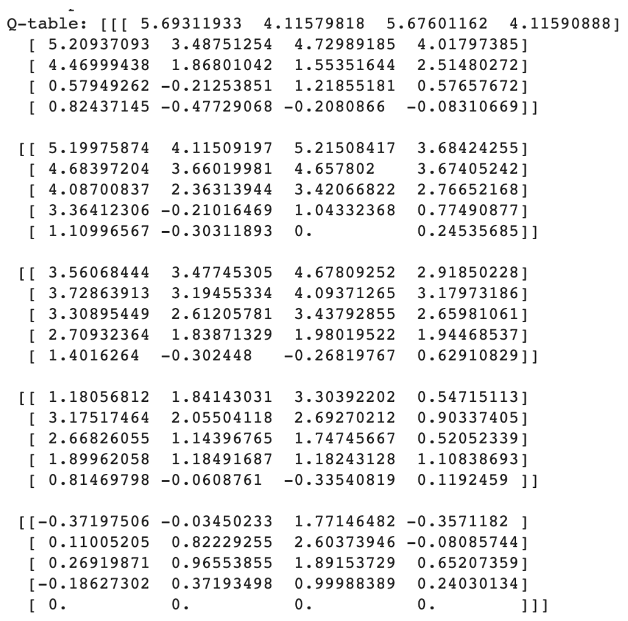
```
[<matplotlib.lines.Line2D at 0x7f2ca501cf98>]
```



The Path followed by the agent is:



The below is the Q Table image :



```
Q-table: [[[ 5.69311933  4.11579818  5.67601162  4.11590888]
  [ 5.20937093  3.48751254  4.72989185  4.01797385]
  [ 4.46999438  1.86801042  1.55351644  2.51480272]
  [ 0.57949262 -0.21253851  1.21855181  0.57657672]
  [ 0.82437145 -0.47729068 -0.2080866  -0.08310669]]

 [[ 5.19975874  4.11509197  5.21508417  3.68424255]
  [ 4.68397204  3.66019981  4.657802    3.67405242]
  [ 4.08700837  2.36313944  3.42066822  2.76652168]
  [ 3.36412306 -0.21016469  1.04332368  0.77490877]
  [ 1.10996567 -0.30311893  0.          0.24535685]]

 [[ 3.56068444  3.47745305  4.67809252  2.91850228]
  [ 3.72863913  3.19455334  4.09371265  3.17973186]
  [ 3.30895449  2.61205781  3.43792855  2.65981061]
  [ 2.70932364  1.83871329  1.98019522  1.94468537]
  [ 1.4016264  -0.302448   -0.26819767  0.62910829]]

 [[ 1.18056812  1.84143031  3.30392202  0.54715113]
  [ 3.17517464  2.05504118  2.69270212  0.90337405]
  [ 2.66826055  1.14396765  1.74745667  0.52052339]
  [ 1.89962058  1.18491687  1.18243128  1.10838693]
  [ 0.81469798 -0.0608761  -0.33540819  0.1192459 ]]

 [[-0.37197506 -0.03450233  1.77146482 -0.3571182 ]
  [ 0.11005205  0.82229255  2.60373946 -0.08085744]
  [ 0.26919871  0.96553855  1.89153729  0.65207359]
  [-0.18627302  0.37193498  0.99988389  0.24030134]
  [ 0.          0.          0.          0.        ]]]
```

6 Conclusion

The agent learns well and performs in a better way for the parameters chosen for the above environment and model.

References

[1] https://www.geeksforgeeks.org/what-is-reinforcement-learning/

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

[3]q=markov+decision+process+images&sxsrf=ACYBGNS5vLDQLJjmWtc3qSMTG3SBKekiOw1575510191859&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjek83qsJ3mAhXBtVkKHV1DDAwQ_AUoAnoECA4QBA&biw=852&bih=827&dpr=1.13#imgrc=qeCP2F8NgTn5DM:

[4] https://towardsdatascience.com/reinforcement-learning-demystified-36c39c11ec14

[5] https://www.geeksforgeeks.org/markov-decision-process/