Gulfam Hussain
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
gulfamhu@buffalo.edu

Machine Learning – Project 4
RL agent grid navigation through Q-Learning (Reinforcement Learning)

Abstract

As part of this project, we need to build a reinforcement learning agent to navigate the 5x5 grid-world environment. The agent has to learn an optimal policy through Q-Learning which will allow it to take actions to reach a goal while avoiding obstacles. The environment and agent should be compatible with OpenAI Gym environment. For reference, the working environment and the framework has been provided for a learning agent.

Introduction:

Reinforcement learning (RL) 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. A reinforcement learning agent learns an optimal policy through Q-Learning to navigate to given grid-world environment and achieves the goal by avoiding the obstacles.

Basic reinforcement is modeled as a Markov decision process:

- a set of environment and agent states, *S*;
- a set of actions, A, of the agent;
- $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability of transition from state (s) to state (s') under action (a).
- $R_a(s, s')$ is the immediate reward after transition from s to s' with action a.
- rules that describe what the agent observes

Environments:

The environment and the framework for a learning agent have been provided. Also, the working model for two agents (random and heuristic agents) have been given for reference.

Implementation and Tasks:

Q-learning (Q stands for quality) 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.

When Q-learning algorithm is performed, a q-table of the shape **[state, state, action]** was used to store and update the values after each episode completed by RL agent. This q-table becomes a reference table for our agent to select the best action based on the q-value for the next episodes going to be performed by the agent.

The agent takes the action based on two environment methods like Exploration and Exploitation. In the exploiting method, the agent selects the action based on the max values available in the q-table for a given state.

In the exploration method, agent is allowed to act randomly instead of selecting the actions based on the max future reward. The exploration/exploitation is balanced by using epsilon(*ε)* value.

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

The above equation shows the q value update and few extra parameters have been used as explained below:

- **Learning rate (lr):** It is 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 (*γ):*** It is a discount factor which is used to balance the immediate and future reward.
- **Reward:** It is the value received after completing a certain action at a given state.
- **Max:** np.max () uses the NumPy library and takes the maximum of the future reward and apply it to the reward for the current state. It impacts the current action by the possible future reward.

Tasks:

Task 1: Implement Policy function

The agent's action selection is modeled as a map called *policy*. The policy map gives the probability of taking action (a) when in state (s).

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

Here, π = *policy*, s = *state* and a= *action.*
When an agent finds a pattern to achieve highest possible reward during the action, it starts following the same after getting trained.

Task 2: Q-table update

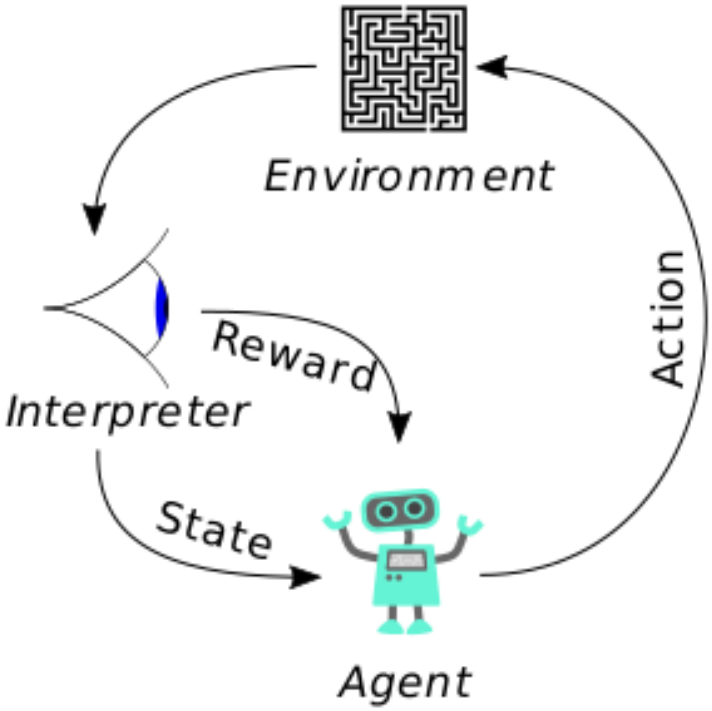
A Q-table consists of state and action as parameters and use to improve the quality of the results for a RL agent while performing the task to attain maximum possible reward. Based on the below equation, we can see that the q-value gets updated after each step performed by the agent and the corresponding new values are used by the agent for the next state until it reaches the goal. Thus, the rewards keep getting increased and in turns it helps to increase the final result too.

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

Task 3: Training Algorithm

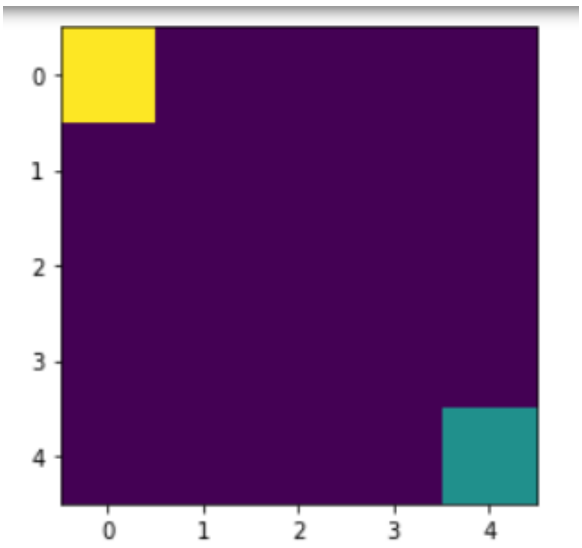
In this task, we train our agent over the range of episodes using learning rate and decay rate. In each episode, we update the state, action, reward and the next state by training the agent using exploration method or using epsilon to let the user finds a pattern after trying all the possible way. When it is not deciding the action randomly, the agent will predict the reward value based on the current state and pick the action that will give the highest reward. An exponential decay is introduced to decrease the number of random action and helps the agent to explore the environment. And the total rewards are calculated after training the agent once it reaches it goal.

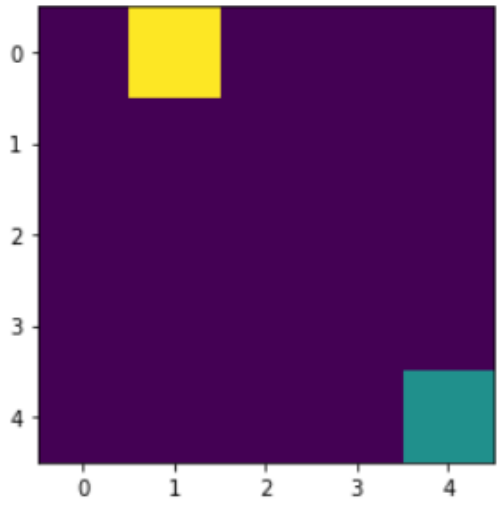
Architecture:

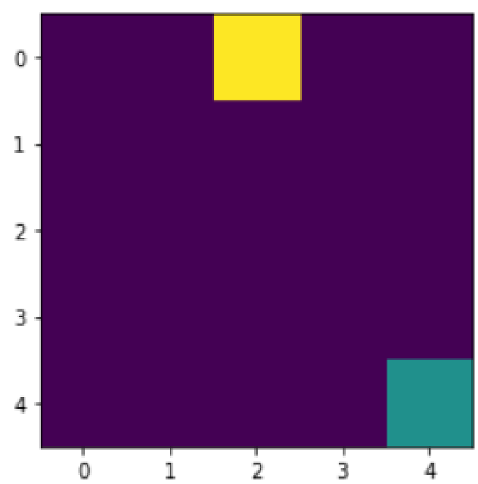
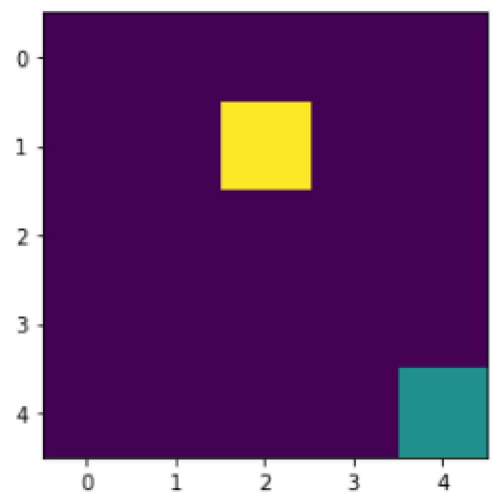
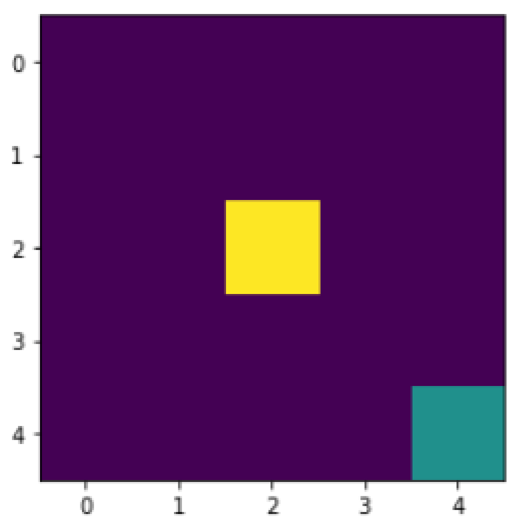
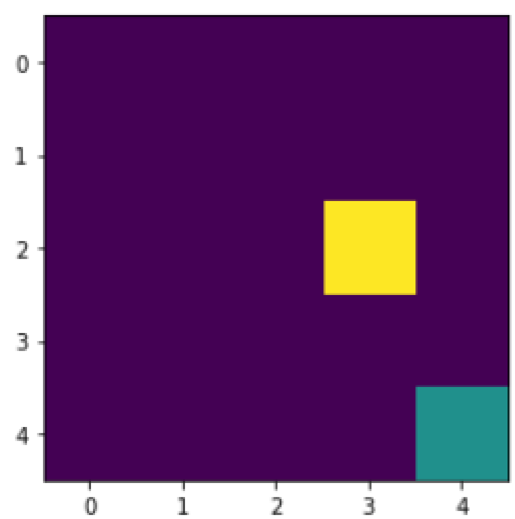
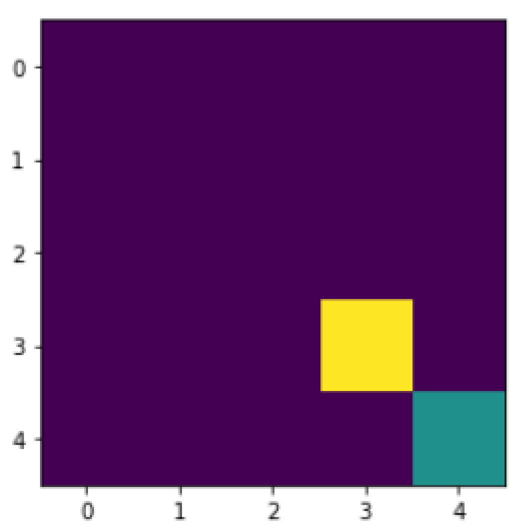
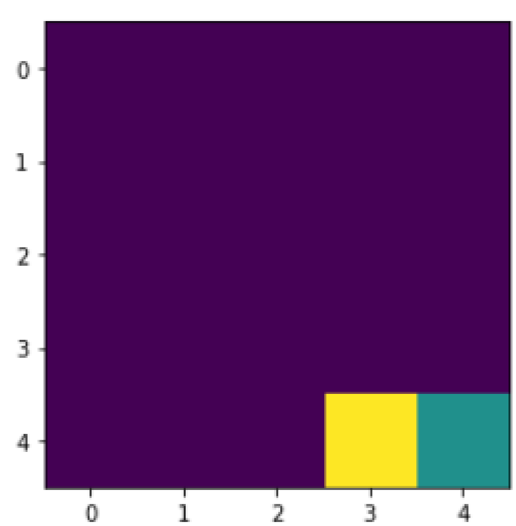


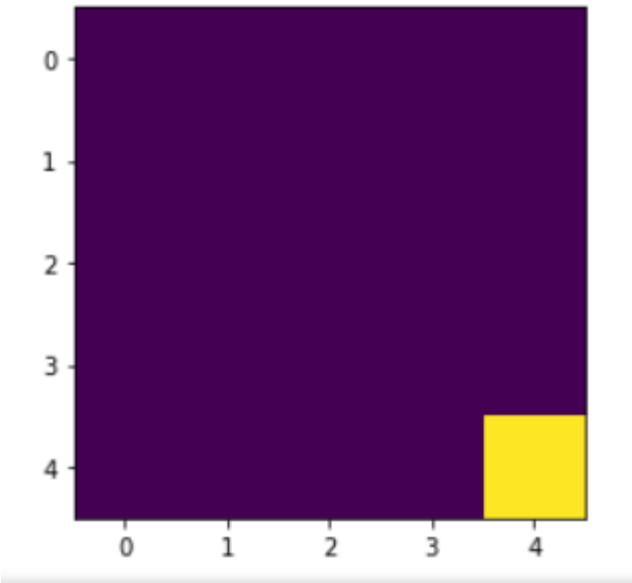
Results:

Agents steps:





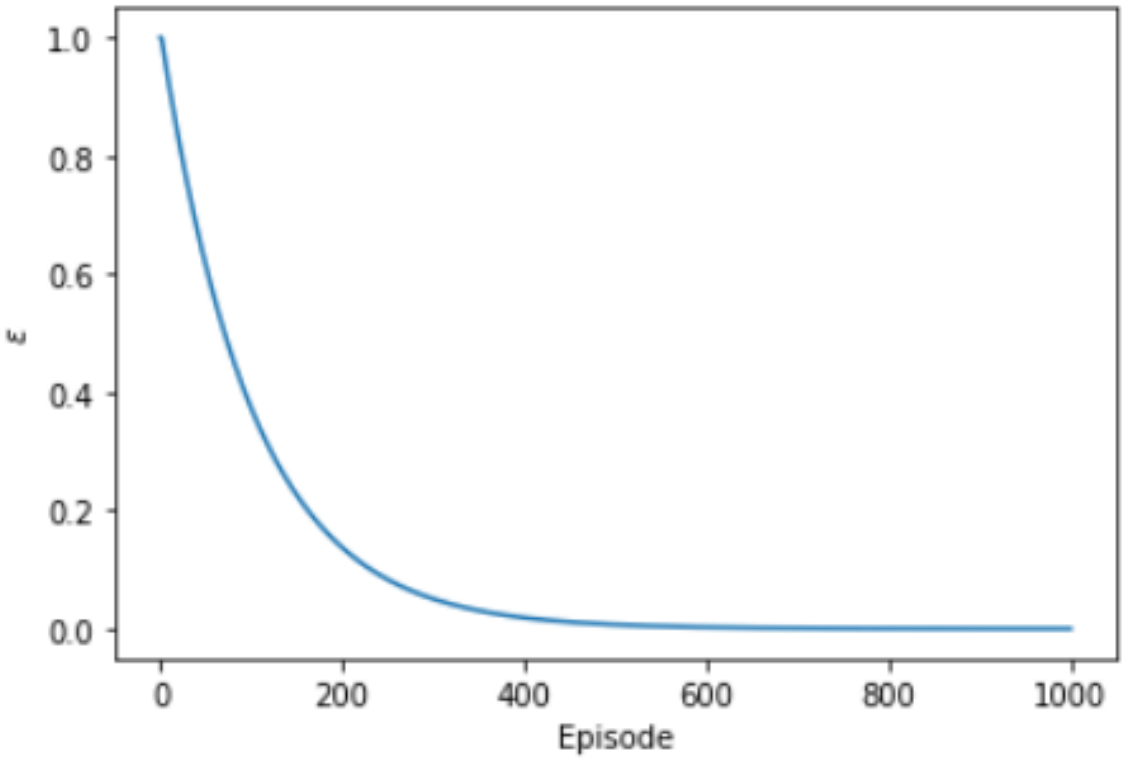

0
1
2
3
4
0 1 2 3 4

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

0
1
2
3
4
0 1 2 3 4

0
1
2
3
4
0 1 2 3 4



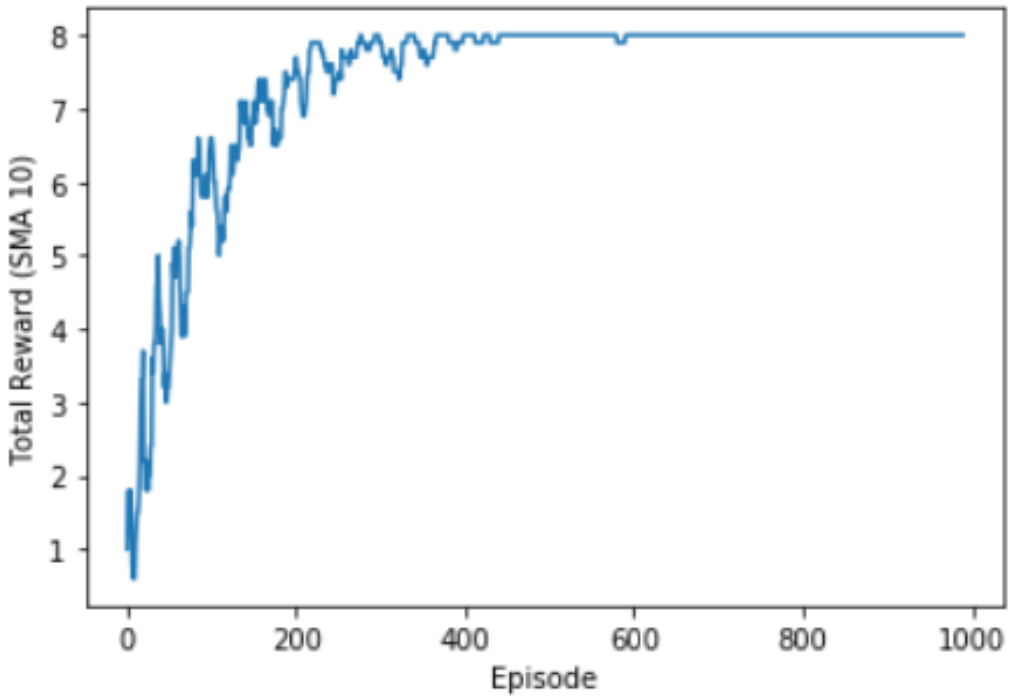
Q-Table:

```
[[[ 2.67639266  2.98003588  5.6953279   2.23378735]
  [ 2.48752708  2.35044669  5.217031    2.61438295]
  [ 4.68559     1.82070409  2.18760678  2.29149696]
  [ 0.43305484 -0.30776975  2.08203554 -0.14061024]
  [ 2.03520339 -0.1720171  -0.28205681 -0.15228442]]

 [[ 2.28161128  0.41369443  1.19038064  0.03243241]
  [ 3.24915562  0.34486112  0.78213259 -0.17910039]
  [ 4.0951      2.09113348  2.30818236  0.57262273]
  [ 2.95909999 -0.02491418  0.66967124  0.04426308]
  [ 1.9043898  -0.10466014 -0.09019    -0.14774788]]

 [[ 2.0143278  -0.01677686  0.52079842 -0.25045412]
  [ 0.96204304 -0.08744806  3.65665156 -0.18564331]
  [ 1.66283743  1.63458809  3.439       0.89869066]
  [ 2.71        0.53850743  1.32809895  0.94531146]
  [ 1.60165418 -0.12659686 -0.06677936 -0.03733869]]

 [[ 0.36829    -0.12484275  1.86418844 -0.30699108]
  [ 0.36829     0.02237126  1.73709805  0.06258609]
  [ 1.84763045  0.05505857  0.42870707 -0.17931876]
  [ 1.9         0.5538431   1.08468965  0.08731528]
  [ 0.94185026 -0.073342   -0.09020667 -0.03122817]]

 [[-0.1729     -0.08209     0.2071     -0.091     ]
  [ 0.         -0.091       0.302761   -0.091     ]
  [-0.39727642 -0.09199147  1.53403702 -0.21451051]
  [-0.24044181  0.00689513  1.          -0.01575414]
  [ 0.          0.          0.          0.        ]]]
```

No of episodes vs epsilon values:



No of episodes vs total rewards per episode:



Conclusion:

This particular project deals with implementing a q-learning model and train our RL agent to achieve maximum reward without any obstacles. A success implementation of training the agent using Q-learning model helped the agent to navigate the environment and finally reach the goal and gets maximum reward on average within a fair number of epochs.

References:

[1] Class slides and project 4 description documents
[2] https://en.wikipedia.org/wiki/Reinforcement_learning
[3] https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56