Dynamic Programming and Reinforcement Learning
Assignment 4

Submitted by:
Abhishek Subramanian Iyer - 2724035
Ander Eguiluz Castañeira - 2739225

INTRODUCTION

In this report, we will be describing the solutions we have achieved for the given Q-learning problem.
This report includes the following :

PART 1 :

In this part of the problem , first we have 5 discrete states and 2 discrete actions, then we extend the problem to that of 10 discrete states while keeping the rewards the same at the end of the chain. The q-value results are stored in a q-table (it is a matrix of size states * actions).

For the implementation we take the following parameters

```
states = np.arange(5) #we have 5 states
alpha = 0.5 #learning rate
epsilon = 1
gamma = 0.9 #discount rate
episodes = 100 #epsilon iterations
iterations = 100 #no.of times we run the program
```

```
states = np.arange(10) #we have 10 states
alpha = 0.5 #learning rate
epsilon = 1
gamma = 0.9 #discount rate
episodes = 100 #epsilon iterations
iterations = 100 #no.of times we run the program
```

We consider the iterations in the form of episodes which contain the number of steps in which the q-table is updated. For this problem, whenever we encounter a state we check if action = 0 (a1), then we traverse the chains and find all the intermediate states where the reward is 0, we keep traversing till we reach the terminal state where the reward is 1. If the given state is 0, then reward = 0.2 if we take action = 1 (a2).

```
q_update = (1 - alpha) * q + alpha * (r + gamma * q_dash) #qlearning formula
```

The above figure displays the q-value implementation. The current q-value (q) is taken from the q-table for current state and action. The maximum next q-value (q_dash) is the max value from the q-table corresponding to the next state. Here, action is chosen using an $\in$-greedy algorithm. This is conducted by taking a random value from the range [0,1] and comparing it to $\in$. Optimal action is chosen from the q-table if said randomly value is > $\in$, otherwise a random action is taken.

For Optimal q values - We changed the value of the parameters to get q values for the following scenarios :

1. We have taken α (learning rate) as 0.5 and we are taking 100 ε - episodes. The γ (discount factor) is given in the question and it is 0.9. We do this for 5 states

```
Episode  70 :
 [[ 6.561   6.1049]
 [ 7.29    5.9049]
 [ 8.1     5.9049]
 [ 9.      5.9049]
 [10.      5.9049]]
Episode:   70, long run average reward:  1.0, current epsilon: -0.02
Episode:   75, long run average reward:  1.0, current epsilon: -0.02
Episode  80 :
 [[ 6.561   6.1049]
 [ 7.29    5.9049]
 [ 8.1     5.9049]
 [ 9.      5.9049]
 [10.      5.9049]]
Episode:   80, long run average reward:  1.0, current epsilon: -0.02
Episode:   85, long run average reward:  1.0, current epsilon: -0.02
Episode  90 :
```

2. We have taken α (learning rate) as 0.5 and we are taking 100 ε - episodes. The γ (discount factor) is given in the question and it is 0.9. We do this for 10 states

```
Episode:   60, long run average reward:  0.2, current epsilon: 0.09
Episode:   65, long run average reward:  0.2, current epsilon: 0.02
Episode  70 :
 [[1.62       2.        ]
 [1.62       1.8       ]
 [1.62       1.8       ]
 [1.61999982 1.8       ]
 [1.60343825 1.79999997]
 [1.21064867 1.79558208]
 [1.19434649 1.67226609]
 [0.         1.69050589]
 [0.         0.        ]
 [0.         0.        ]]
Episode:   70, long run average reward:  0.2, current epsilon: -0.02
Episode:   75, long run average reward:  0.2, current epsilon: -0.02
```

3. We have taken α (learning rate) as 0.5 and we are taking 1000ε - episodes. The γ (discount factor) is given in the question and it is 0.9. We do this for 5 states

```
Episode:  660, long run average reward:  1.0, current epsilon: 0.01
Episode:  665, long run average reward:  1.0, current epsilon: 0.00
Episode  670 :
 [[ 6.561   6.1049]
 [ 7.29    5.9049]
 [ 8.1     5.9049]
 [ 9.      5.9049]
 [10.      5.9049]]
Episode:  670, long run average reward:  1.0, current epsilon: -0.00

Episode:  675, long run average reward:  1.0, current epsilon: -0.00
Episode  680 :
 [[ 6.561   6.1049]
 [ 7.29    5.9049]
 [ 8.1     5.9049]
 [ 9.      5.9049]
 [10.      5.9049]]
Episode:  680, long run average reward:  1.0, current epsilon: -0.00
Episode:  685, long run average reward:  1.0, current epsilon: -0.00
```

4. We have taken α (learning rate) as 0.5 and we are taking 1000ε - episodes. The γ (discount factor) is given in the question and it is 0.9. We do this for 10 states

```
Episode:  660, long run average reward:  1.0, current epsilon: 0.01
Episode:  665, long run average reward:  1.0, current epsilon: 0.00
Episode  670 :
 [[ 3.87420489  3.6867844 ]
 [ 4.3046721   3.4867844 ]
 [ 4.782969    3.4867844 ]
 [ 5.31441     3.4867844 ]
 [ 5.9049      3.4867844 ]
 [ 6.561       3.4867844 ]
 [ 7.29        3.4867844 ]
 [ 8.1         3.4867844 ]
 [ 9.          3.4867844 ]
 [10.          3.4867844 ]]
Episode:  670, long run average reward:  1.0, current epsilon: -0.00
Episode:  675, long run average reward:  1.0, current epsilon: -0.00
Episode  680 :
 [[ 3.87420489  3.6867844 ]
 [ 4.3046721   3.4867844 ]
 [ 4.782969    3.4867844 ]
```

We can see that for the same learning rate even if we change the number of states from 5 to 10, both the cases converge at the same state.

PART 2:

In this part, we have to solve the chain problem using function approximators for $\gamma = 0.9$ and 10 states.

Background

For this part, the already implemented DeeR library (https://github.com/VinF/deer) has been used.

To model the problem in this library, we only need to update the init(), reset() and act() methods. The transitions within the chain, which have been already explained in Part I are modeled in act() through a list of 10 states [0-9] and a set of 2 actions. The state is represented as 1 scalar and it is initialized to 0 in init() and reset to 0 in reset().

Now we need to set the appropriate hyperparameters: learning rate (alpha), discount rate (gamma) and epsilon for our specific case.

To run the simulation, first we need to install a particular set of library-versions due to the compatibility of Python, keras and tensorflow versions. Once we have represented our state and actions

I Provide illustrations of the solutions of your optimal q-values

For this problem, we run a simulation of 50 epochs and 1000 steps per epoch.

> *Disclaimer*
> Even though the optimal q-values for part 2 should converge to the values obtained in part 1, the q-values for action 2 diverge significantly from those in part 1. This means there is some unidentified error in our implementation of the system.

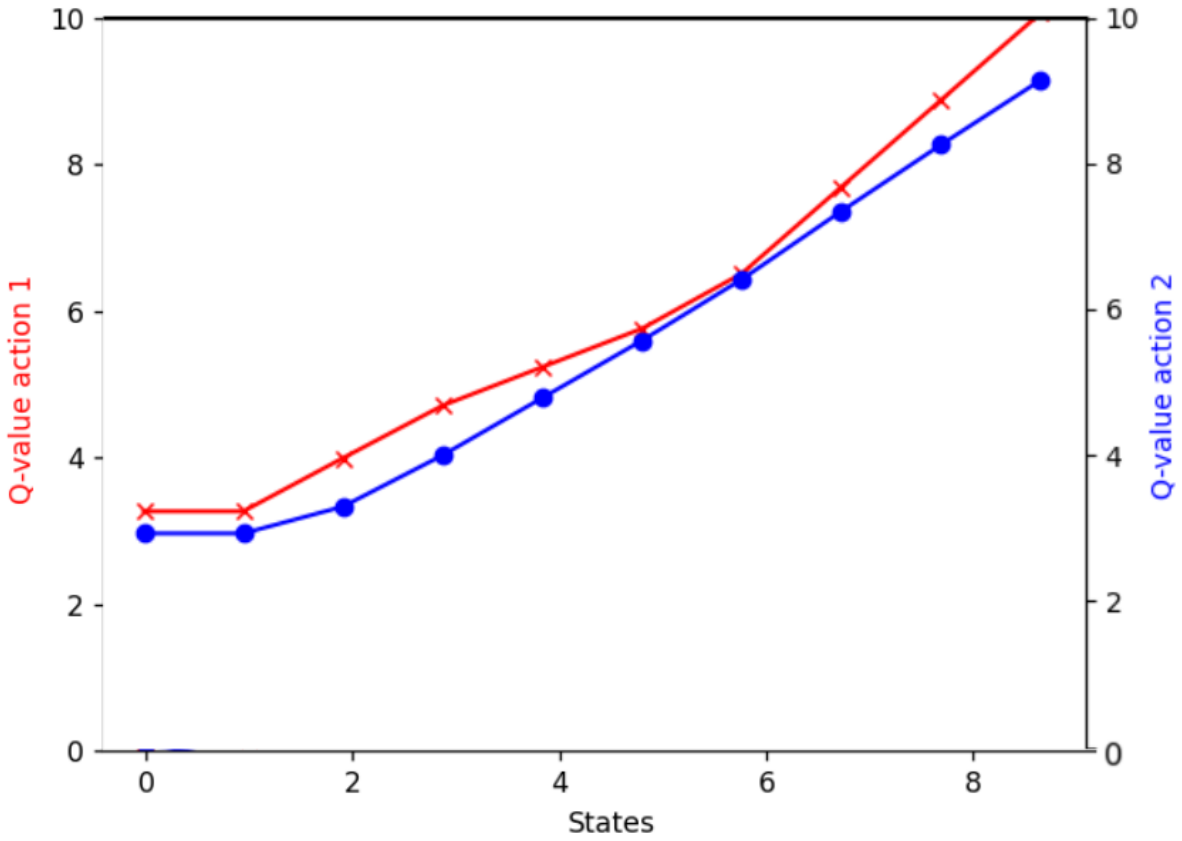
After 25 epochs with α=0.005, the V value is 8.43204 and the q-values are:

```
Average (on the epoch) training loss: 0.11454288430085587
Episode average V value: 8.432046388864517
[3.2692215 2.8368216]
[3.6481407 3.2558477]
[4.096663  3.8170815]
[4.542356  4.3943686]
[4.9880486 4.971656 ]
[5.794619 5.65257 ]
[6.7324815 6.3711853]
[7.6703444 7.0898   ]
[8.608208  7.8084154]
[9.546072 8.527031]
```

After 50 epochs with α=0.005, the V value is 9.1312 and the q-values are:

```
Average (on the epoch) training loss: 0.114072392459417 26
Episode average V value: 9.132119899988174
[3.2309866 2.9276776]
[3.2309866 2.9276776]
[3.9613595 3.295936 ]
[4.68118  4.000203]
[5.208613 4.78643 ]
[5.736046  5.5726566]
[6.489894  6.4100904]
[7.6785064 7.345852 ]
[8.869477 8.256586]
[10.062857  9.141769]
epoch 50:
```

The resulting plot is:



II Discuss the hyper-parameters and the convergence

- Learning rate (α):

Using the learning rate, we can control how quickly the model adapts to the problem. Given the smaller changes made to the weights each update, a smaller learning rate requires more training epochs, whereas larger learning rates result in more rapid changes with fewer training epochs.

We need a learning rate that is not too large, because it can cause the model to converge too quickly to a suboptimal solution, and also one that is not too small, since it can cause the process to get stuck.

For our example, trying a bigger alpha (α=0.1) resulted in a really fast but suboptimal convergence of q-values.

- Epsilon (ϵ):

Within the exploitation vs exploration tradeoff, epsilon refers to the probability of choosing exploration. In our case, we initialize epsilon to 1, fostering the exploration and then leaving it

decaying through the *epsilon-decay* parameter progressively to 0 to boost the exploitation of the best solution.

- Convergence:

After running a certain number of epochs, depending on the hyper-parameters, the optimal q-values should converge to those already seen in part 1.