REINFORCEMENT LEARNING APPLIED TO DISCRETE AND CONTINUOUS SPACES

Divya Balasubramanian Reema Chhatbar

- **Basic Task**

1. Define a Domain and Task

Reinforcement learning teaches an agent on how to choose an action from its action space within an environment aiming to maximize rewards over time. The environment is a framework in which an agent is present to search for an optimal path. When the environment has finite action choices it is said to be a discrete environment whereas in the continuous environment the space of possible actions senses infinitely. In this project, for the basic task, we will be working on a grid-world problem of what we call as the 'Modified cliff walking' which is inspired by the 'Cliff walking' problem in [1]. In our problem, we have modified the cliff walking environment into a 10 x 10 grid with multiple cliff locations. The aim is to create a Q-learning algorithm where the agent is expected to find the optimal path between the starting point (S) and the end goal (G) as seen in Figure 1.

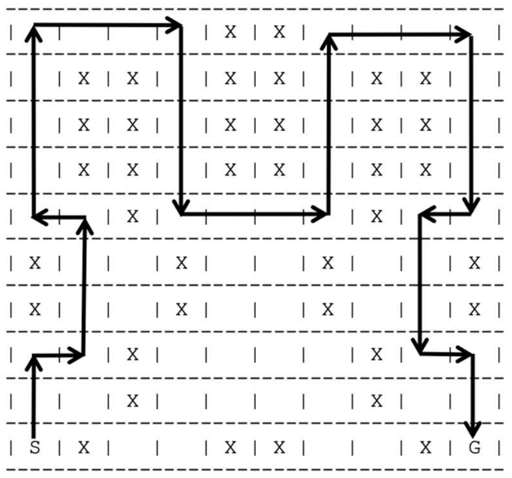


Figure 1a: Modified cliff walking environment (optimal path)

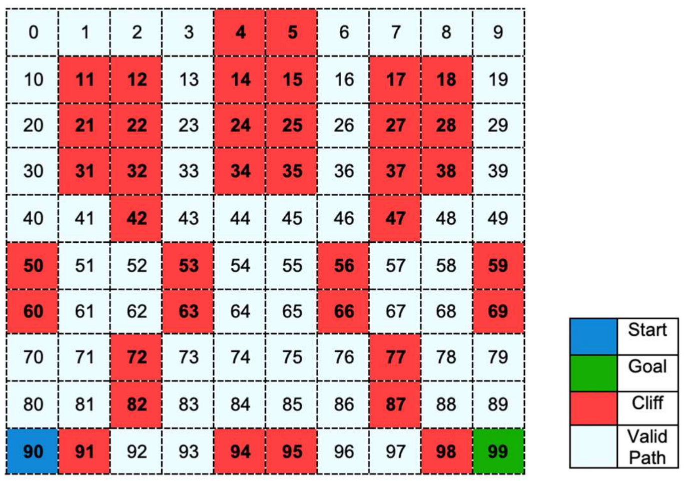


Figure 1b: Modified cliff walking environment (explained)

As opposed to this simple problem that can be solved by Q-learning, we go on to explore a continuous environment for the advanced task where we will be applying Deep Q networks to solve the game. For this, we have chosen the 'MountainCar-v0' environment from OpenAI gym [2]. This will be discussed in Section 10.

2. Define the State Transition and Reward Function

2.1. State Transition:

In the Modified cliff walking problem, the agent moves through this discrete environment with the ability to take one of four different actions namely up, down, left, or right in each state. In this problem, we apply the off-policy temporal difference approach also known as Q-learning [1] to find the optimal path from the start to the end goal. Q-learning is applied based on the formula shown in Figure 2 where:

Q - Q value, S - State, A - Action, R - Reward, α (Alpha) - Learning rate, γ (Gamma) - Discount factor, t (subscript) - time of any given state

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Figure 2: 'One step Q learning' defined in [1]

An agent following Q-learning works by taking an action based on the maximum Q value of a [state, action] pair and updating the Q value based on the action taken. This continues till the values converge to obtain the optimal path.

From the grid-world environment in Figure 1, we can see the starting position is always at (9,0). As this is a 10 x 10 grid, the agent has multiple transitions possible from each state, hence we provide the initial state transition as an example below.

Initial State	Action	Next State
(9,0)	Up	(8,0)
(9,0)	Right	(9,1)

Table 1: Initial state transition of the agent

2.2. Reward function:

As this problem has been adapted from the original cliff walking problem, the rewards have been designed similarly. Here, the agent receives a reward of -100 if it chooses to walk into any of the cliffs (marked as X in Figure 1a), -1 for every additional step it takes before it reaches the goal and 100 when the goal (G) is reached. The reward function that defines the value of an agent moving to a new state is defined by:

rt+1 = r(st, at)

The reward matrix encoded with values will be discussed in section 4.2.

3. Choose a Policy

We have tested the algorithm using two different approaches. We first created a random agent which does not require learning. This agent does not follow any policy and chooses a random action in any given state. It is useful to compare this with Q-learning to analyse the improvement using learning.

For Q-learning, the agent follows the ***ε*** greedy policy. Here ***ε*** stands for epsilon which greedily explores or exploits the environment. By exploration, the agent explores the possible actions it can take from the current state and chooses a random action however exploitation is the process where it would take an action based on the largest value in the Q table. The behaviours will occur by ***ε*** and 1-***ε*** respectively. Setting ***ε*** as 1 in the initial instance allows the agent to explore the environment in the first few episodes as the Q table is initialised to zero. We however will be *decaying* the value of ***ε*** using a decay rate until a threshold is reached i.e. ***ε***min in order to avoid the problem of local minima which would arise on setting a null minimum value. The aim of the decay rate reducing the value of ***ε*** is to incentivise the agent to learn the wider environment in fewer random actions whilst also maximising the learning from the Q table.

4. Graphical Representation of the Problem and R Matrix

4.1. Graphical Representation:

We have designed a compact example of the grid-world problem in a graphical format as seen in Figure 3. In this case, there are an exponential number of possible combinations of states and actions which would not be realistic to show in this graph. Since the index has been designed as shown in Figure 1b, the 'Start (S)' state is at s(9,0) or 90 as represented in the graph. The 'Goal (G)' is S(99,9) or 99 which is the final state obtained by taking a 'down' final action.

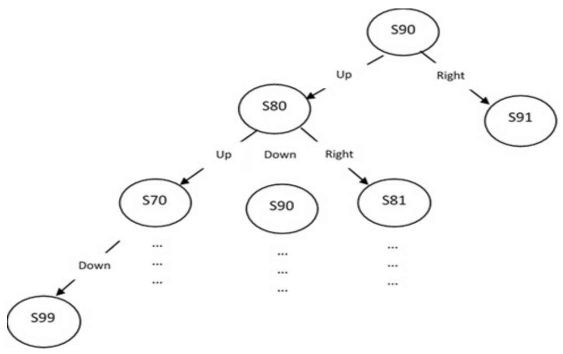


Figure 3: Graphical representation of Modified Cliff Walking problem

4.2. R Matrix:

The R matrix defines the rewards an agent receives for all possible actions it can take from a given state. We have initialised the R Matrix to assign rewards based on the action taken, as mentioned in section 2.2.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	-1	-1	-1	-1	-100	-100	-1	-1	-1	-1
1	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1
2	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1
3	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1
4	-1	-1	-100	-1	-1	-1	-1	-100	-1	-1
5	-100	-1	-1	-100	-1	-1	-100	-1	-1	-100
6	-100	-1	-1	-100	-1	-1	-100	-1	-1	-100
7	-1	-1	-100	-1	-1	-1	-1	-100	-1	-1
8	-1	-1	-100	-1	-1	-1	-1	-100	-1	-1
9	-1	-100	-1	-1	-100	-100	-1	-1	-100	-1

Table 2: R Matrix for the Modified Cliff walking environment

5. Set the Parameter Values for Q-Learning

Based on the formula in figure 2, we have defined the following hyperparameters for the initial run.

- Alpha = 0.9
- Gamma = 0.9
- Epsilon = 1
- Epsilon (min) = 0.05
- Epsilon decay rate = 0.995

We have selected the above default options to first gather an understanding of how the model functions with these assigned hyperparameters. Since the value for Alpha, the learning, is set at 0.9, we would have to incentivise the agent to learn quickly by having a high Gamma value, discount factor, which decreases the reward with every additional step taken.

6. How the Q-matrix is Updated in a Learning Episode?

In this section, we will show a hand worked example of two episodes of how the Q-matrix is updated using initial R-matrix in Table 2.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0

Table 3: Initial Q-Matrix

The following is a pseudo code for how the Q-Matrix is updated for our environment:

1. Define the initial state (9,0), s0
2. Define possible actions from this state s0 (a0) = (8,0) – Up, (9,1) – Right.
3. Select an action (a0) based on ε greedy policy
4. The action a0 defines the next state (s1)
5. Update Q(s0,a0)
6. Break

In this case, point 1 & 2 of the pseudo code will remain the same in all the episodes as the initial state is always at (9,0).
As there are an extremely high number of possible calculations to reach the end-goal, **we provide a toy example** of how the Q-Matrix is updated in this environment. In this example, we consider α = 0.9 and γ = 0.9.

Episode 1:
For episode 1, the agent starts at s0, following the ε greedy policy we assume in this case it explores the environment and so moves right to (9,1) which is a terminal state as it is the cliff.
(9,0) -> (9,1)
Q[s0,a0]new = Q[s0,a0]old + α * (Rt+1+ γ * max(Q[s1,A1]) - Q[s0,a0]old)
Q[9,0]new = Q[9,0]old + 0.9 * (-100 + 0.9 * (max Q(9,1)) – Q[9,0]old)
Q[9,0]new = 0 + 0.9 *(-100 + 0.9*(0) – 0)
Q[9,0]new = -90

Based on this episode we see the Q-Matrix is updated as in Table 4 (highlighted in yellow).

Episode 2:
For this episode, the agent starts at s0, follows exploitation to take the first two actions and then follows exploration to take the final action as seen below.
(9,0) -> (8,0) -> (8,1) -> (8,2) where (8,2) is the terminal state as it is the cliff.
Q[9,0]new = Q[9,0]old + 0.9 * (Rt+1 + 0.9 * max ((9,1), (8,0)) – Q[9,0]old)
Q[9,0]new = -90 + 0.9 *(-1 + 0.9*(0) – (-90)) = -9.9
Similarly,
Q[8,0]new = 0 + 0.9*(-1+ 0.9*(0)-0) = -0.9
Q[8,1]new = 0 + 0.9*(-1+ 0.9*(0)-0) = -0.9
Q[8,2]new = 0 + 0.9*(-100 + 0.9*(0) – 0) = -90
The updated Q-Matrix following the two episodes is displayed in Table 4 (Episode 1 highlighted in yellow and Episode 2 highlighted in green).

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	-0.9	-90	0	0	0	0	0	0	0	0
8	-0.9	0	0	0	0	0	0	0	0	0
9	-9.9 (-90)	0	0	0	0	0	0	0	0	0

Table 4: Q-Matrix after Episode 1 and Episode 2

7. Represent Performances vs Episodes

The Q-learning algorithm with learning rate and discount factor set to 0.9 was iterated over 50000 episodes. We set epsilon at 1 with a minimum value of 0.05 and decay rate of 0.995. As explained in section 2.2, a reward of 100 is assigned for the goal state. We consider the algorithm has converged when it identifies the optimal path to reach the goal (Figure 1a and Appendix A). We can see in the graphs below that the algorithm continues to learn over increasing number of episodes. In Figure 4a, we see the cumulative reward initially dips to a negative value, this is because if the agent explores the environment it is likely to 'fall' into the cliff thereby getting a large negative reward of -100 each time. As the range of values are quite high in this figure, we decided to take a running mean over steps of 500 episodes each time to identify noticeable patterns. In Figure 4b depicting the running mean, we can see although the reward in the initial episodes was a high negative value, it increases with increasing episodes as the agent learns the optimal path using exploitation alongside.

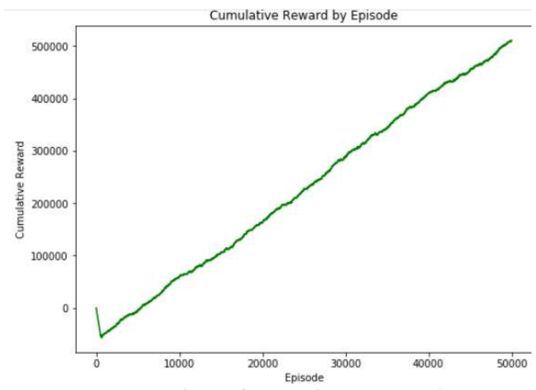


Figure 4a: Number of episodes vs cumulative reward

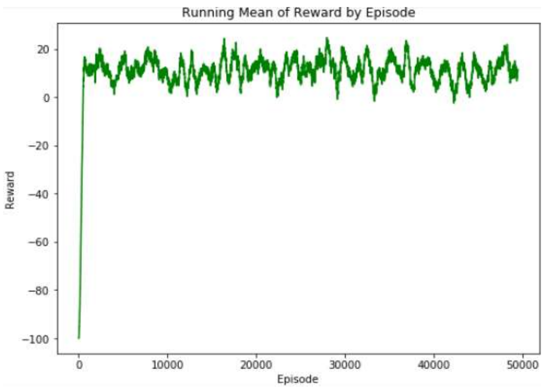


Figure 4b: Number of episodes vs reward

8. Varying the Parameters

We first created a random agent which was compared with the performance of a Q learning agent as seen in section 8.1. Once it was established that the Q-learning agent performs better, we applied grid search for hyperparameter optimisation on the discount factor (gamma), learning rate (alpha) and epsilon decay rate, which is discussed in subsequent sub-sections.

8.1. Random Agent vs Q-Learning Agent

As mentioned in section 3, a random agent was created to compare the performance with the Q-learning agent. As seen in Figure 5, the random agent followed completely random actions with no learning thus having a very high negative cumulative reward and a standard running mean of -100 as it always fell in the cliff thus not converging or reaching the goal using the optimal path. However, as discussed in section 7, the agent following Q-learning learns the environment better with increasing number of episodes.

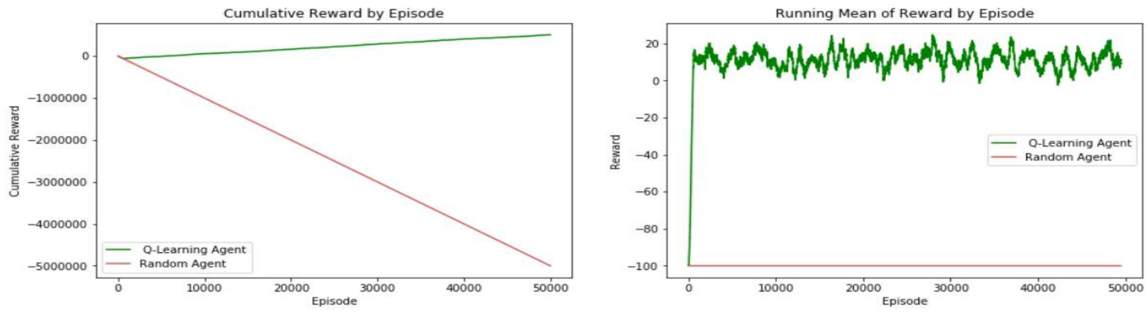


Figure 5: Comparison of performance between random agent and Q-learning agent

8.2. Discount Factor

As per [1], the discount factor or gamma (γ) defines the present value of future rewards. In other words, gamma can be considered as a penalty added to the reward of a future state for every additional step the agent takes to reach that state. We ran this experiment over five different values of gamma while maintaining $\alpha = 0.8$ and $\varepsilon = 1$. As seen in Figure 6a, $\gamma = 0.9$ shows the best result in terms of cumulative reward.

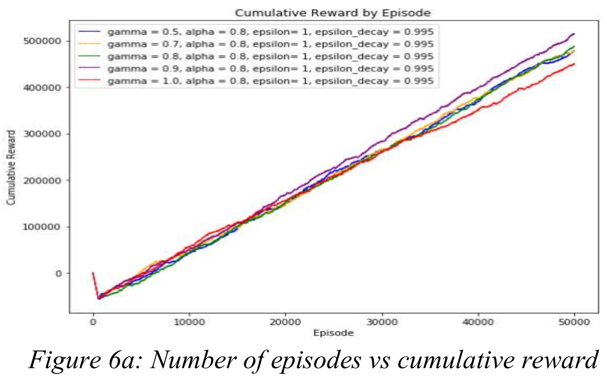


Figure 6a: Number of episodes vs cumulative reward

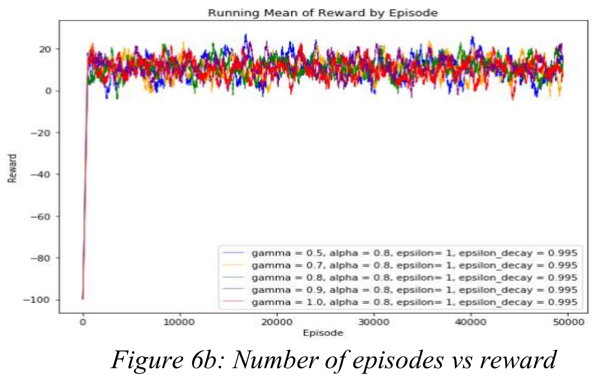


Figure 6b: Number of episodes vs reward

Since the running mean for the different gamma values has overlapping trends, we decided to take the average of the last 1000 episodes of each run to identify the best performing value of γ. This mainly shows the value of the reward achieved consistently at the end of the process although the algorithm uses the ε greedy policy and therefore converges towards an optimal policy. As seen in Table 6, $\gamma = 0.9$ produces the highest average reward thus coinciding with results noted in Figure 6a.

Gamma	Mean Rewards for Last 1000 Episodes
0.5	14.4
0.7	5.0
0.8	16.6
0.9	17.2
1	9.2

Table 5: Mean rewards for last 1000 episodes of different gamma values

8.3. Learning Rate

The learning rate or alpha (***α***) defines how quickly the Q-learning agent learns the optimal path. We note that in discrete spaces such as our environment which have only four possible actions the learning rate probably plays a lesser role than in continuous spaces with infinite possible actions. However, we ran a grid search over five values of ***α*** to identify the one that produces the largest mean reward over the last 1000 episodes. We have maintained ***γ*** = 0.9 (obtained from section 8.2) and ***ε*** = 1. We found that ***α*** = 0.9 shows the highest mean reward over the last 1000 episodes (Figure 7a, 7b and Table 6).

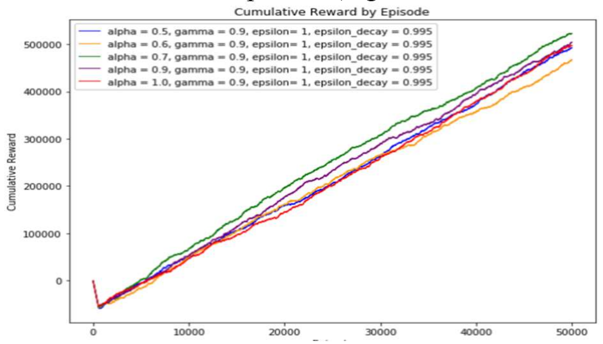


Figure 7a: Number of episodes vs cumulative reward

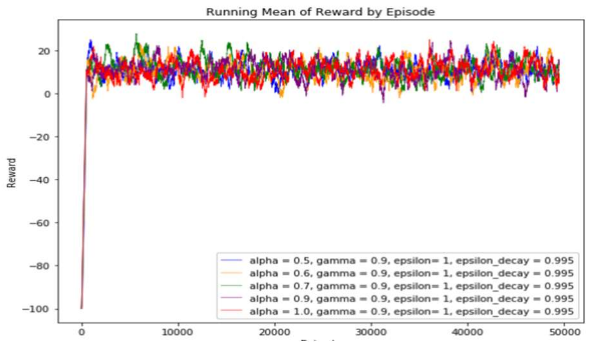


Figure 7b: Number of episodes vs reward values

Alpha	Mean Rewards for Last 1000 Episodes
0.5	10.2
0.6	11.6
0.7	8.0
0.9	12.4
1	9.0

Table 6: Mean rewards for last 1000 episodes of different alpha values

8.4. Different Policies

The ***ε*** greedy policy largely depends on the decay rate and minimum epsilon value threshold. Varying the decay rate would impact the epsilon value in the ***ε*** greedy policy and thereby Q-learning agent's ability to find the optimal path quickly. We have experimented using four different decay rates to modify the policy maintaining ***α*** = 0.9 and ***γ*** = 0.9 (obtained from sections 8.2 and 8.3). Figures 8a, 8b and Table 7 show us that decay rate = 0.995 produces the best results in finding the optimal path.

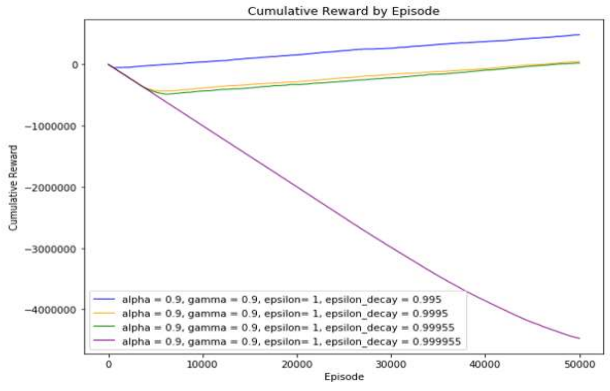


Figure 8a: Number of episodes vs cumulative reward

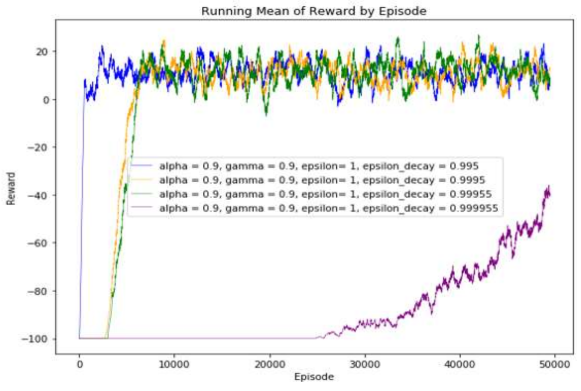


Figure 8b: Number of episodes vs reward

Decay Rate	Mean Rewards for last 1000 Episodes
0.995000	10.6
0.999500	10.4
0.999550	8.8
0.999955	-39.6

Table 7: Mean rewards for last 1000 episodes of different decay rate values

9. Analysis of Results

On training using default values, we see the Q-learning agent is indeed able to identify the optimal path to the goal state. However, due to the large number of cliffs in this Modified cliff walking environment and following the $\boldsymbol{\varepsilon}$ greedy policy, we see that the agent takes longer identify the optimal path. It is interesting to note from Figure 8a and 8b, that the highest value of decay rate (0.99955) shows a poor performance. In this case, we see the running mean shows a standard -100 reward until about 25000 episodes. This could be because the $\boldsymbol{\varepsilon}$ value is decayed at a very quick pace due to which the agent depends on exploiting the values of the Q-Matrix which is initialised with zeroes rather than exploring the environment. Due to this it does not begin learning until it crosses around 25000 episodes after which there is a steady increase in values although much lower than the other decay rates. The lowest value of 0.995 produces the best results with the mean reward being 10.6 and we can see it begins learning from as early as around 2000 episodes in comparison to other values. This could be due to the higher opportunity for exploration due to the lower decay rate.

The discount factor or $\boldsymbol{\gamma}$ impacts the finite Markov Decision Process (MDP) in an environment involving finite state and action spaces [1]. This is because with a low value of $\boldsymbol{\gamma}$, the agent becomes short-sighted and does not consider the future steps as important. In our case, we note that an undiscounted reward where $\boldsymbol{\gamma} = 1$ still produces reasonable results although not the best one compared to other values (Table 5). This could be because there is a finite goal with standard actions with bounded rewards. Yamaguchi et al (2012) [3] argue that the discount factor may not be relevant in this learning framework when compared to animal behaviour where the stimulus received is high-dimensional sensory input over standard state-definition with a reward or learning. This an important factor that can perhaps be considered if this problem is applied in a real-life scenario involving some impulsive decisions.

Whilst a high learning rate or $\boldsymbol{\alpha}$ is considered to be beneficial, we note the learning rate when set at $\boldsymbol{\alpha} = 1$ produced a 9.0 mean reward over 1000 episodes which was low compared to the highest value of 12.4 at $\boldsymbol{\alpha} = 0.9$. We think this could be a consequence of working in combination with $\boldsymbol{\varepsilon}$ being set to 1 in this experiment due to which the agent learns the random exploratory actions taken initially thereby taking longer to identify the optimal path.

Although we have performed the popular method of hyperparameter optimization using grid search where in the Q-learning agent finds the optimal path, we did not achieve the optimal path reward of 62 [Appendix A] in any of the combinations as yet. This can be due to varying one parameter while maintaining others at a constant value. Since there are multiple locations of cliff in our modified environment, perhaps in future work random search [4] can be applied for hyperparameter optimization. In [4], Bergstra et al (2012) describe the importance or weight of different hyperparameters vary based on the dataset thereby highlighting that a random search of the environment may identify better results. In future work we would also be interested in applying other policies such as Boltzmann policy which is applied on discrete spaces or on-policy methods such as Sarsa to find the optimal path for comparison.

- **Advanced Reinforcement Learning Techniques**

10. Introduction

In this section we will be extending our study to a continuous environment 'MountainCar-v0' from OpenAI gym [2]. While we have worked with Q-learning so far for the discrete Modified cliff walking environment, we will be adapting Deep Q Networks (DQN) which use functional approximators to solve this continuous problem within a large state space such as this. This concept was first introduced by Mnih et al (2013) [5] to play Atari games which have high dimensional sensory input. DQN avoid storing the infinite states and actions a Q-learning agent would need to otherwise store and consider for updating its Q values. In DQN, we would input the state to the neural networks which would then output the Q values of the possible actions as output.

10.1. Define a Domain and Task

As mentioned in section 10, we have chosen to work on the MountainCar-v0 problem. This a classic problem that entails making an under-powered car at the foothill to reach the flag at the top of a mountain. Each step the car takes is given a -1 reward and the goal is to touch the flag on top of the mountain which is at 0.5 position [6]. Once the car reaches the flag at the top of the mountain, the episode ends. The problem is considered "solved" if the agent receives an average reward of -110 over 100 consecutive trials [6]. Although simple, it is a tricky problem as the agent would have to observe two inputs namely position and velocity, and take one of three actions, i.e. push left, no push and push right, to achieve the goal.

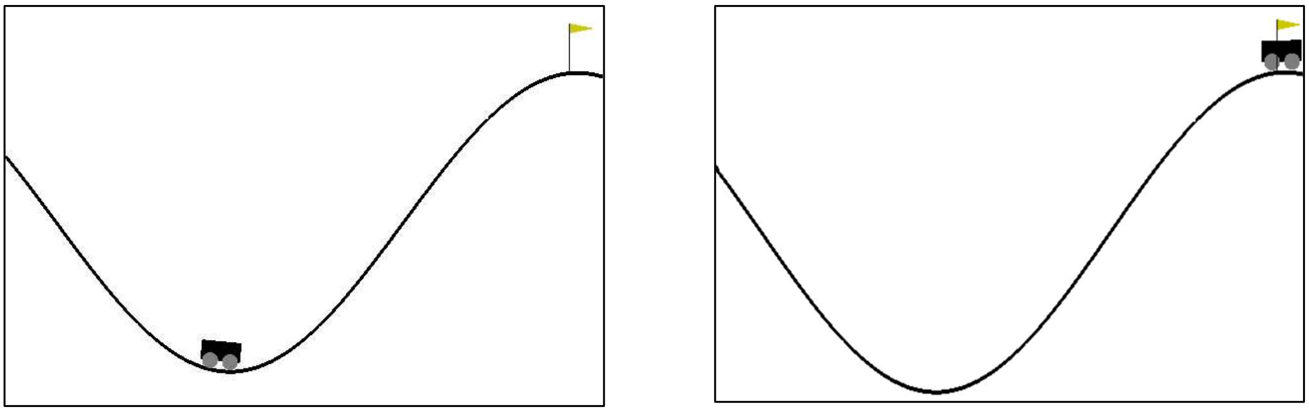


Figure 9: MountainCar-v0 environment from OpenAI gym [2]

10.2. DQN Agent

DQN uses neural networks as functional approximators to find the Q values that would enable the agent to progress in the environment and converge, i.e. gaining an expected average reward continuously over a certain number of episodes. In this continuous environment, the agent considers sequences of observations rather than a single observation as it would define the direction of movement of the car and thereby decide the appropriate action to be taken. As stated in [5], this gives rise to a finite Markov decision process where each sequence of observations is a considered as a distinct state.

In order to achieve this, we modify the Bellman equation (revisited below from Section 2.1) to match the requirements of the DQN.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

Figure 10: Bellman equation revisited [1]

Assuming that s is the sequence of observations and a is the sequence of actions, Q*(s,a) represents the maximum expected return at time step t. To enable to the DQN to identify the optimal value of Q*(s', a') for the next time step, the equation is modified as below:

$$Q^*(s,a) = y_i = \mathbb{E}_{s'\sim\varepsilon}[r + \gamma max_{a'} Q^*(s',a')|s,a]$$

Figure 11: Modified equation from [5]

Where we have the loss of the gradient with θ as weights of the neural network defined as:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

Figure 12: Loss function from [5]

Using this concept, we applied a DQN with 1 hidden layer of 100 neurons with a ReLU activation function. We gave the 2 state observations as input and 3 expected actions are the expected output. The model was trained using an Adam optimizer with a learning rate of 0.001, discount factor of 0.97 and a mean squared error loss. In order to allow an opportunity to explore the environment alongside exploitation, we once again considered the ***ε*** greedy policy as with the basic task. The decay rate was maintained at 0.99 decaying the ***ε*** value up to a minimum threshold of 0.1. We ran the experiment over 400 episodes and the following results were observed:

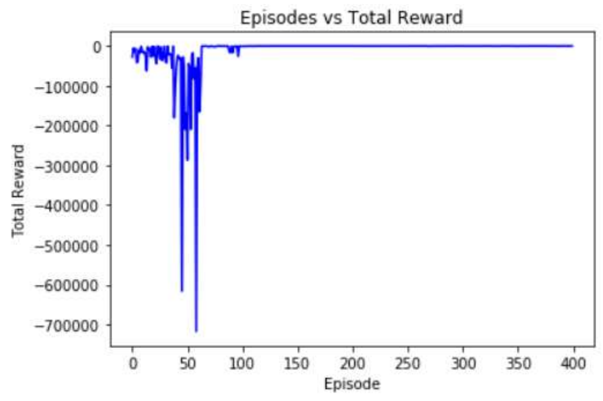


Figure 13a: Episodes vs total reward

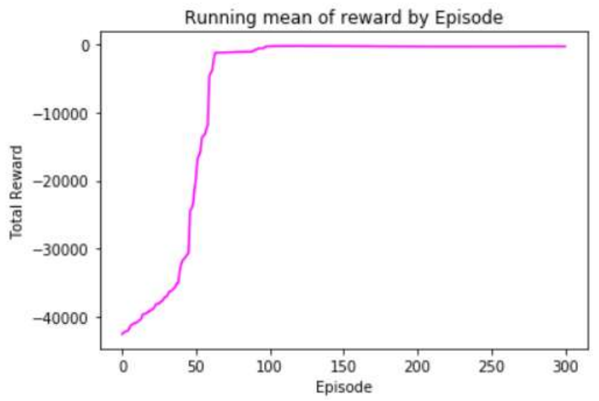


Figure 13b: Running mean of total reward by episode

As the results in Figure 13a are with a range of high values varying across episodes, we decided to calculate the running mean across the total reward in steps of 100 episodes as seen in Figure 13b.

10.3. Experience Replay

The concept of experience replay was first used in combination with DQNs in [5] where the agent's experience in each time-step over several episodes are stored in a replay memory. In our algorithm, we have set the replay memory to a size of 10000 which accumulates experiences as the agent passes the different states in the environment. Once the memory touches the maximum limit, it acts as a sliding window to add a new experience while removing the oldest one. During training, it is possible to randomly sample some experiences from this memory for the agent to learn. In our case, we have set the random sample to 50 episodes. This helps to avoid highly correlated samples from interfering in the learning process.

Below we have created a pseudo code to reiterate how the experience replay algorithm plays into a DQN (the amendments for Double DQN which will be discussed in section 10.4 has also been added):

```
1. Initialise ReplayBuffer (experience replay) of maxlen (memory size)
2. Initialise Q values with random weights (and the target network for Double DQN agent)
3. Set the number of episodes
4. For episodes in the range of episodes in step 3,
  4.1. Reset the environment (at the start of every episode)
  4.2. Initialise total rewards to zero (this will be cumulative sum as the episode progresses)
  4.3. While episode is not done
    4.3.1. The agent undertakes an action using the $\varepsilon$ greedy policy.
    4.3.2. Execute the action and observe the next state and reward
    4.3.3. Train the agent on the network
```

    4.3.4. Update the total reward based on the rewards obtained
    4.3.5. The experience replay memory is updated
    4.3.6. Where DoubleDQN is used update the target network weights
5. Break

We used similar values of hyperparameters as the DQN Agent in section 10.2, however this time we initialised experience replay to support the agent's learning.

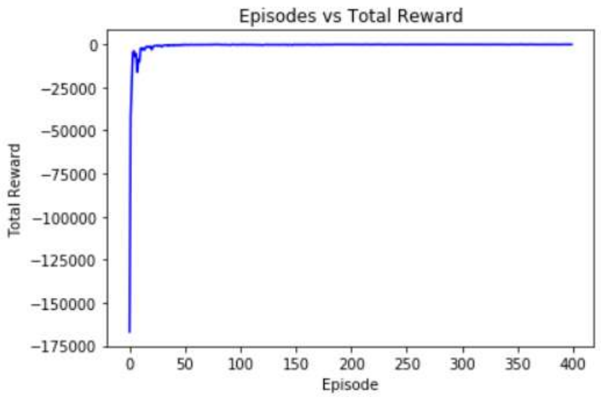


Figure 14a: Episodes vs total reward

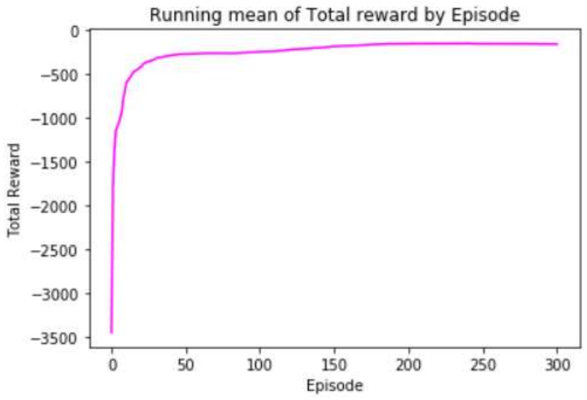


Figure 14b: Running mean of total reward by episode

As we can see from Figure 14a and 14b, the performance of the agent has largely enhanced with the introduction of experience replay.

10.4. Double DQN (DDQN) Agent

Hasselt et al (2015) [7] have discussed about Q-learning falling into the trap of an overestimation bias. In continuous environments where the agent may follow a stochastic process depending on the policy, the overestimation bias will largely interfere with the agent's learning process as the maximum Q value will be selected to decipher the next state (based on the Bellman equation). The solution proposed in [7] involves using two networks, one local network that updates the weights based on the loss calculated while the second target network, whose weights are not always updated, decides the target Q value thus forming the DDQN.
We decided to apply the DDQN agent with experience replay to the MountainCar-v0 environment to observe the results. We maintained the same values for the hyperparameters, including the number of episodes, to make the results comparable with the previous agents applied. The results are shown in Figure 15a and 15b.

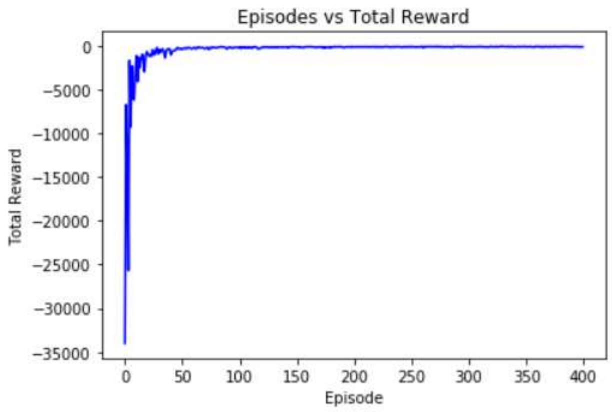


Figure 15a: Episodes vs total reward

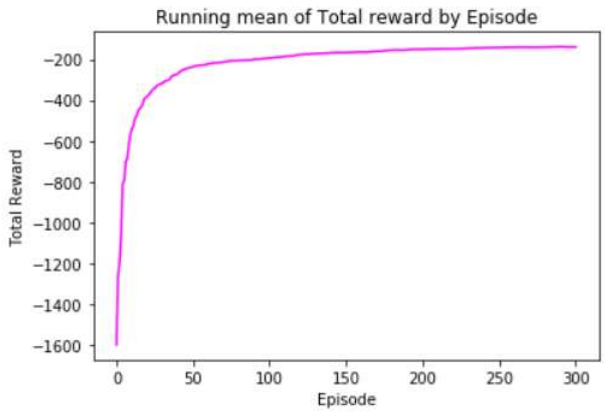


Figure 15b: Running mean of total reward by episode

10.5. Comparing Double DQN and DQN in Alternate Runs

We experimented applying the DDQN and DQN in alternate runs to produce a direct comparison of the running mean over 100 episodes. We maintained the same hyperparameters and each run had 400 episodes. The even numbered runs were trained with the DDQN agent and odd numbered runs with DQN Agent. The results are shared in Figure 16.

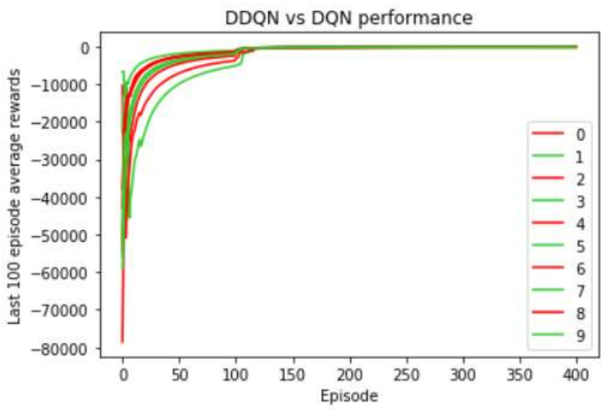


Figure 16: Running mean over 100 episodes

11. Quantitative and Qualitative Analysis of Results of Advanced Reinforcement Learning Techniques

In this section we will be performing a critical evaluation of the results obtained from section 10.

The Mountain car environment is relatable to a real-life scenario where there are infinite possible states which are influenced by factors such as position and velocity that encourage us to take actions accordingly to reach a goal. We applied a simple DQN agent with fully connected layers to learn and solve this environment using ε greedy policy. From Figure 13a, we can see the agent makes a number of random choices prior to reaching the goal in each episode thereby accumulating a total reward of large negative values (in the range of -710,000 per episode) much after the first 50 episodes. Here, the agent learns the environment without access to experience replay which evidently slows down the rate of learning. We note that over the 400 episodes, the agent does not reach the optimal expected average reward as needed to "solve" this problem.

We then introduced experience replay to enhance the capability of the DQN agent. Experience replay has direct biological plausibility where the agent uses memory to decide on which action it should take next. This is very similar to human nature to perform an action through memory. Here, we have stored 10000 experiences in memory, which in essence can be considered as a hyperparameter when optimising the model. The introduction of experience replay to the agent shows us a drastic improvement in the pace of learning. From Figure 14a, we can decipher that the agent spends first 20 episodes exploring the environment which is validated by the large negative values of total reward. As we drilled into the data [Appendix B], we note that it in fact begins to move towards convergence around the 51st episode after which the average reward moves closer to the required average of -110 over 100 episodes [6]. The reward in this environment is -1 for every additional step the agent takes to reach the goal. This has been maintained as such to make our results comparable with [6]. However, it is worth noting that having a standard reward of -1 does not incentivise the agent to take an alternate action to improve performance.

The learning rate plays an import role in influencing the agent's ability to learn the environment. In this case, the learning rate also works in congruence with the loss with respect to updating the weights of the network. It is a hyperparameter that is normally set between 0 and 1. While a small learning rate might require larger number of training epochs, a large learning rate may make the model converge too quickly to a suboptimal level. In our case, it has been set to 0.001 which is a low rate comparatively and may have perhaps played a causal role in the rate of update of weights in the network and thereby the agent's performance.

However, we see a better performance for the same learning rate when the DDQN agent with experience replay is trained. From Figure 15a, we can see the range of values of total reward has largely shrunken in comparison with the performance of the DQN agent in Figure 14a. The fact that a target network is now trained to identify the target Q values alongside an 'online' network that computes the loss and updates the weights of the network has helped the agent learn more quickly although the learning rate is still set to 0.001.
We also ran the DDQN and DQN agents over 10 different runs of 400 episodes each alternating between each other to produce a direct comparison as in Figure 16. We can see that in most runs DDQN tends to perform better than DQN.

While both our DQN and DDQN agents converged to reach the goal, we note they did not reach the -110 average reward over 100 episodes. Towards the end of 400 episodes, we see the DQN agent converges towards an average reward of ~ -150 while the DDQN agent towards ~ -130 average reward. This validates that the agents would perhaps need to be trained with increased number of episodes or with tuned hyperparameters to optimise the model to the solved state.

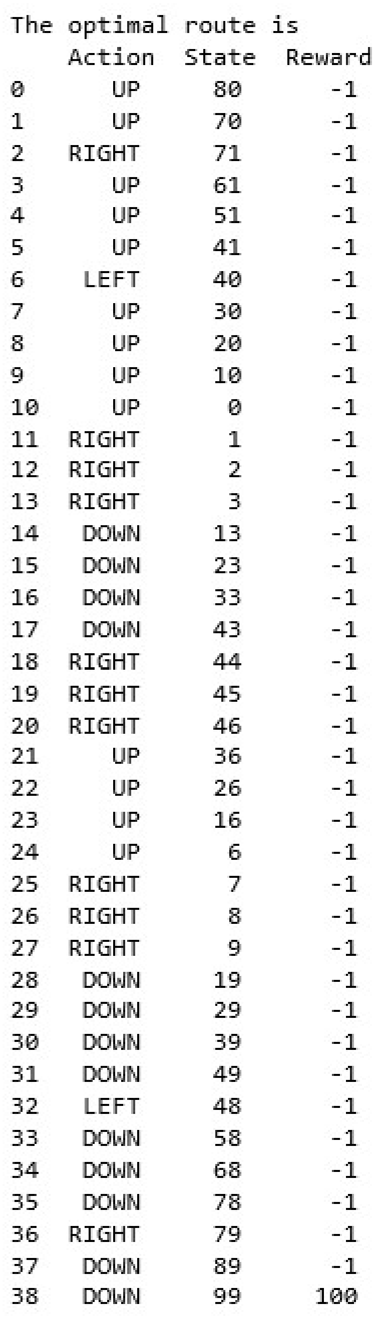
In future work, we would be interested to apply hyperparameter optimisation through grid search or random search which may help to identify the optimal parameters to improve the performance of the agent. A further advancement to this problem would be to use consecutive image frames from the game as in [5] as inputs to the model.

REFERENCES:

[1] Sutton, R.S. & Barto, A.G. 1998, Reinforcement learning: an introduction, MIT Press, Cambridge, Mass.
[2]OpenAI, MountainCar-v0. Available at: https://gym.openai.com/envs/MountainCar-v0/ (Accessed: 15 April 2020)
[3] Yamaguchi, Y. & Sakai, Y. 2012, "Reinforcement learning for discounted values often loses the goal in the application to animal learning", Neural Networks, vol. 35, pp. 88-91.
[4] Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(Feb), 281-305.
[5] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. 2013, "Playing Atari with Deep Reinforcement Learning", .
[6] Github (2020), MountainCar v0 Available at: https://github.com/openai/gym/wiki/MountainCar-v0 (Accessed: 30 April 2020)
[7] van Hasselt, H., Guez, A. & Silver, D. 2015, "Deep Reinforcement Learning with Double Q-learning", .
[8] Source code for advanced task adapted from https://github.com/the-computer-scientist/OpenAIGym/blob/master/DeepQNetworksInOpenAIGym.ipynb

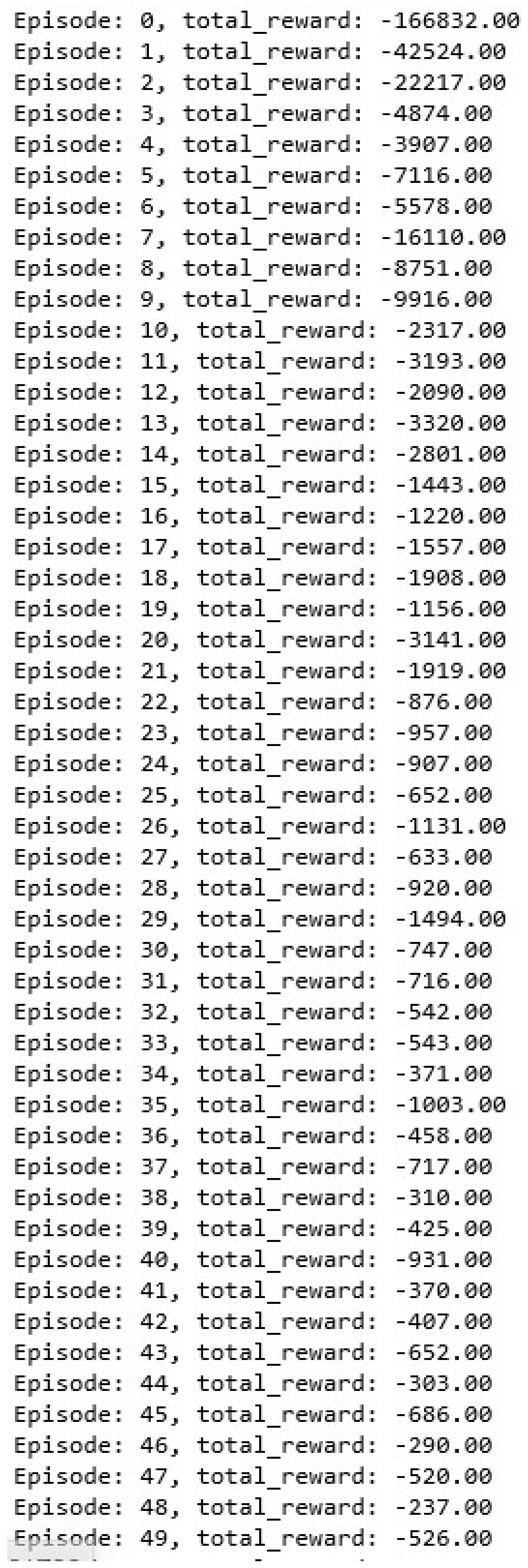
APPENDIX:

APPENDIX A – Optimal path for the Modified Cliff walk obtained by the Q-learning agent (from our code)

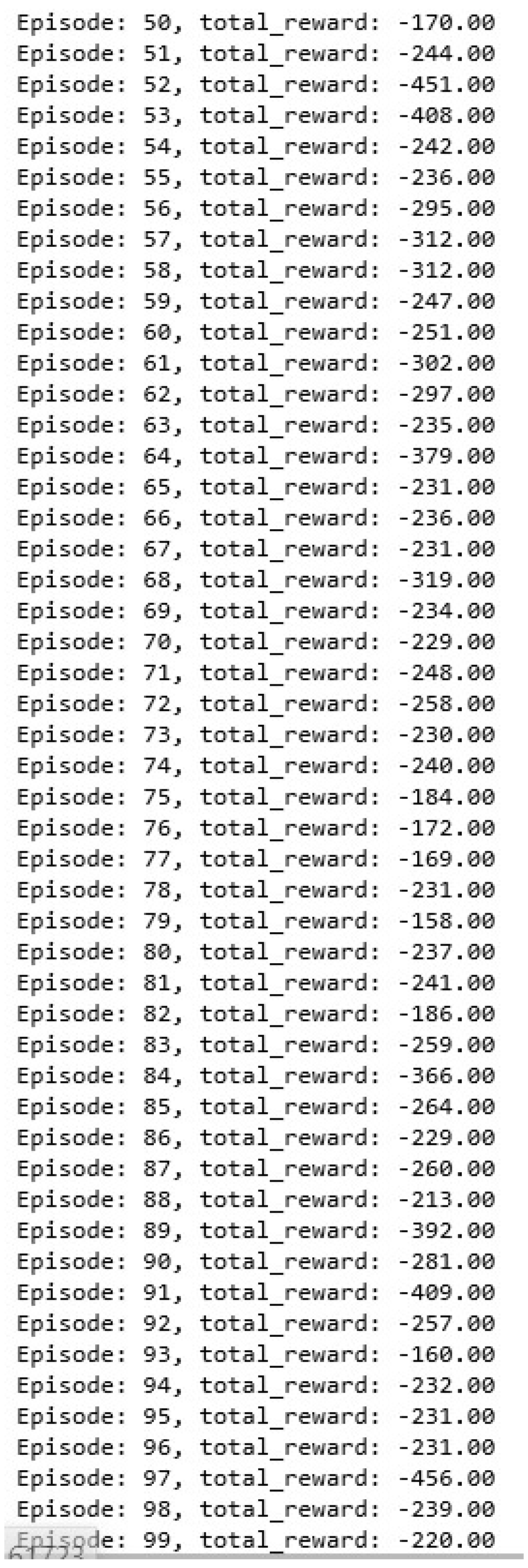


```
The optimal route is
    Action  State  Reward
0       UP     80      -1
1       UP     70      -1
2    RIGHT     71      -1
3       UP     61      -1
4       UP     51      -1
5       UP     41      -1
6     LEFT     40      -1
7       UP     30      -1
8       UP     20      -1
9       UP     10      -1
10      UP      0      -1
11   RIGHT      1      -1
12   RIGHT      2      -1
13   RIGHT      3      -1
14    DOWN     13      -1
15    DOWN     23      -1
16    DOWN     33      -1
17    DOWN     43      -1
18   RIGHT     44      -1
19   RIGHT     45      -1
20   RIGHT     46      -1
21      UP     36      -1
22      UP     26      -1
23      UP     16      -1
24      UP      6      -1
25   RIGHT      7      -1
26   RIGHT      8      -1
27   RIGHT      9      -1
28    DOWN     19      -1
29    DOWN     29      -1
30    DOWN     39      -1
31    DOWN     49      -1
32    LEFT     48      -1
33    DOWN     58      -1
34    DOWN     68      -1
35    DOWN     78      -1
36   RIGHT     79      -1
37    DOWN     89      -1
38    DOWN     99     100
```

APPENDIX B – Sample of transition from high negative values towards convergence in MountainCar v0 DQN model



```
Episode: 0, total_reward: -166832.00
Episode: 1, total_reward: -42524.00
Episode: 2, total_reward: -22217.00
Episode: 3, total_reward: -4874.00
Episode: 4, total_reward: -3907.00
Episode: 5, total_reward: -7116.00
Episode: 6, total_reward: -5578.00
Episode: 7, total_reward: -16110.00
Episode: 8, total_reward: -8751.00
Episode: 9, total_reward: -9916.00
Episode: 10, total_reward: -2317.00
Episode: 11, total_reward: -3193.00
Episode: 12, total_reward: -2090.00
Episode: 13, total_reward: -3320.00
Episode: 14, total_reward: -2801.00
Episode: 15, total_reward: -1443.00
Episode: 16, total_reward: -1220.00
Episode: 17, total_reward: -1557.00
Episode: 18, total_reward: -1908.00
Episode: 19, total_reward: -1156.00
Episode: 20, total_reward: -3141.00
Episode: 21, total_reward: -1919.00
Episode: 22, total_reward: -876.00
Episode: 23, total_reward: -957.00
Episode: 24, total_reward: -907.00
Episode: 25, total_reward: -652.00
Episode: 26, total_reward: -1131.00
Episode: 27, total_reward: -633.00
Episode: 28, total_reward: -920.00
Episode: 29, total_reward: -1494.00
Episode: 30, total_reward: -747.00
Episode: 31, total_reward: -716.00
Episode: 32, total_reward: -542.00
Episode: 33, total_reward: -543.00
Episode: 34, total_reward: -371.00
Episode: 35, total_reward: -1003.00
Episode: 36, total_reward: -458.00
Episode: 37, total_reward: -717.00
Episode: 38, total_reward: -310.00
Episode: 39, total_reward: -425.00
Episode: 40, total_reward: -931.00
Episode: 41, total_reward: -370.00
Episode: 42, total_reward: -407.00
Episode: 43, total_reward: -652.00
Episode: 44, total_reward: -303.00
Episode: 45, total_reward: -686.00
Episode: 46, total_reward: -290.00
Episode: 47, total_reward: -520.00
Episode: 48, total_reward: -237.00
Episode: 49, total_reward: -526.00
```



```
Episode: 50, total_reward: -170.00
Episode: 51, total_reward: -244.00
Episode: 52, total_reward: -451.00
Episode: 53, total_reward: -408.00
Episode: 54, total_reward: -242.00
Episode: 55, total_reward: -236.00
Episode: 56, total_reward: -295.00
Episode: 57, total_reward: -312.00
Episode: 58, total_reward: -312.00
Episode: 59, total_reward: -247.00
Episode: 60, total_reward: -251.00
Episode: 61, total_reward: -302.00
Episode: 62, total_reward: -297.00
Episode: 63, total_reward: -235.00
Episode: 64, total_reward: -379.00
Episode: 65, total_reward: -231.00
Episode: 66, total_reward: -236.00
Episode: 67, total_reward: -231.00
Episode: 68, total_reward: -319.00
Episode: 69, total_reward: -234.00
Episode: 70, total_reward: -229.00
Episode: 71, total_reward: -248.00
Episode: 72, total_reward: -258.00
Episode: 73, total_reward: -230.00
Episode: 74, total_reward: -240.00
Episode: 75, total_reward: -184.00
Episode: 76, total_reward: -172.00
Episode: 77, total_reward: -169.00
Episode: 78, total_reward: -231.00
Episode: 79, total_reward: -158.00
Episode: 80, total_reward: -237.00
Episode: 81, total_reward: -241.00
Episode: 82, total_reward: -186.00
Episode: 83, total_reward: -259.00
Episode: 84, total_reward: -366.00
Episode: 85, total_reward: -264.00
Episode: 86, total_reward: -229.00
Episode: 87, total_reward: -260.00
Episode: 88, total_reward: -213.00
Episode: 89, total_reward: -392.00
Episode: 90, total_reward: -281.00
Episode: 91, total_reward: -409.00
Episode: 92, total_reward: -257.00
Episode: 93, total_reward: -160.00
Episode: 94, total_reward: -232.00
Episode: 95, total_reward: -231.00
Episode: 96, total_reward: -231.00
Episode: 97, total_reward: -456.00
Episode: 98, total_reward: -239.00
Episode: 99, total_reward: -220.00
```