Deep Learning Coursework: Optimization

Vithurshan Vijayachandran and Hisho Rajanathan

Basic Case

Domain and Task

This paper aims to solve a simple grid problem by implementing the Q-Learning reinforcement learning algorithm. The agent will begin in the top left of the grid (environment) and the aim of this agent is to find the shortest possible path to the bottom right of the grid. The agent will need to navigate across the environment by moving down, left and right.

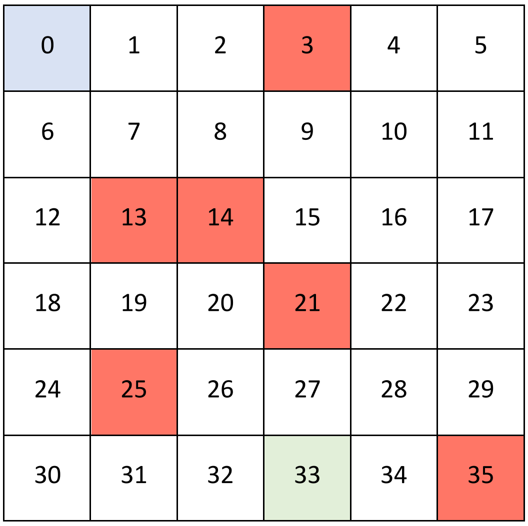

Figure 1 Representation of grid problem

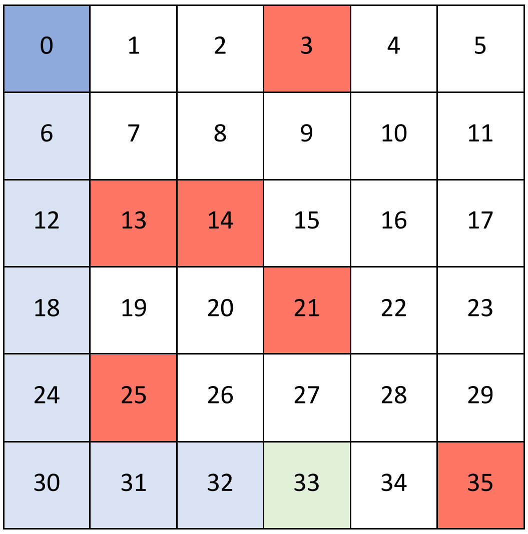

Figure 2 Shortest Possible Path

This problem can be replicated for a number of real-world problems such as the game of checkers or chess, whereby each piece will need to navigate across the board while avoiding certain parts of the board. This is a very simplified version of the checkers or chess game as we have kept fixed points on the board which the agent should not enter.

State Transition Function & Reward Function

The transition function is denoted by $s_{t+1} = \delta(s_t, a_t)$, where by s_{t+1} denotes the state the agent will move to from state s_t, using an action a_t. The agent will be deployed in State 0 - this is where the state transition grid will begin. The goal of the agent is to reach State 33, by avoiding States 3, 13, 14, 21, 25 and 35. Each action will represent a move (action) in the environment moving from one state to another. The agent will be able to move downwards, to the left or right from a state. The agent has the ability to move in and out of states coloured red, however the agent will be given a negative reward for entering these states. The shortest possible route for the agent will be to go through 0 -> 6 -> 12 -> 18 -> 24 -> 30 -> 31 - > 32 - > 33. 33 has been chosen as the reward state as the route stated above will be the only shortest possible route and this would be the most logical path. Figure 2 shows the shortest possible path for the agent to travel from State 0 to State 33.

State (s_t)	Action (a_t)	New State (s_{t+1})
0	Right (move to state 1)	1
0	Down (move to state 6)	6

Figure 3 Possible transitions from the beginning state 0

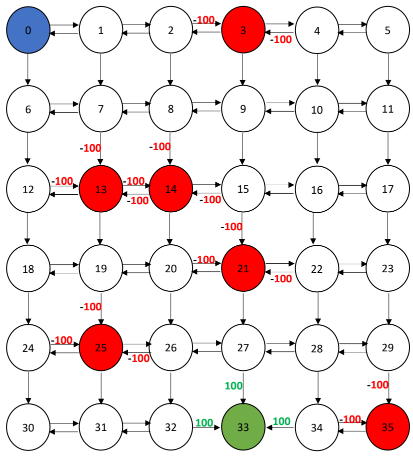

Figure 4 Graphical Representation of reward scores transitioning between states

A reward function is used to incentivise the agent to reach the end state, by avoiding states that give the agent a negative reward. The reward function is denoted by $r_{t+1} = r(s_t, a_t)$, similar to the state transition, where r_{t+1} is the reward for moving to the next state from state s_t, using an action a_t.

Deep Learning Coursework: Optimization

Vithurshan Vijayachandran and Hisho Rajanathan

If the agent moves into a state coloured red i.e. State 3, 13, 14, 21, 25 or 35, negative rewards will be accumulated by the agent. If the agent moves into the reward State, 33, the agent will accumulate a positive reward. No rewards will be accumulated by the agent from moving to any other state and the main objective is to find the shortest possible path to the reward state.

State	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
0		0					0																													
1	0		0					0																												
2		0		-100					0																											
3			0		0					0																										
4				-100		0					0																									
5					0							0																								
6								0					0																							
7							0		0					-100																						
8								0		0					-100																					
9									0		0					0																				
10										0		0					0																			
11											0							0																		
12														-100					0																	
13													0		-100					0																
14														-100		0					0															
15															-100		0					-100														
16																0		0					0													
17																	0							0												
18																				0					0											
19																			0		0					-100										
20																				0		-100					0									
21																					0		0					0								
22																						-100		0					0							
23																							0							0						
24																										-100					0					
25																									0		0					0				
26																										-100		0					0			
27																											0		0					100		
28																												0		0					0	
29																													0							-100
30																																0				
31																															0		0			
32																																0		100		
33																																		100		
34																																		100		-100
35																																			0	

Figure 5 R - Matrix

There are states within the environment where it is impossible for the agent to move to - for example, moving directly from State 0 to State 33 which can be seen in Figure 4 and Figure 5. This has also been included in the code to ensure the agent does not make these impossible transitions. A reward function of 0 is given to the agent if a possible State transition is made, -100 if the agent transitions into a 'trap' state, and 100 if the agent transitions into the reward State. All the 'trap' states have the same negative reward of -100. Figure 5, the R – Matrix (reward matrix), shows the immediate reward received after executing an action and which actions can be executed from a state.

Policy

In order to solve this basic grid problem, an Epsilon greedy policy has been implemented. The policy is denoted by $\pi(s_t) = a_t$. The Epsilon value, ε, has been initialised by generating a random number between 0 and 1. The epsilon greedy search allows us to set a trade-off between exploitation and exploration; an extreme value of ε = 1 shows a random policy and ε = 0 will always choose a greedy policy. If the random number generated is greater than the epsilon value that we have pre-set, the algorithm will be exploiting. All the next possible states will be found, and the agent will choose the state which has the highest reward assigned to it and transition to this state.

A decay factor, λ, was applied to the epsilon greedy policy. The agent at the beginning will be exploring when it has little knowledge of the environment, however as time goes on the agent will require less exploration and should be exploiting the environment. The decay factor will be updated at the end of each episode.

Parameter values for Q-Learning

Part of the Q – Learning algorithm, there are certain hyper parameters that will be required as seen in Figure 6.

Parameter Name	Parameter Symbol	Parameter Value
Learning Rate	α	0.9
Discount Factor	γ	0.8
Epsilon	ε	0.9
Decay Factor	λ	0.9998
Policy	π	Epsilon Greedy

Figure 6 Initial Hyper-parameters for Q-Learning

The learning rate,α controls how fast the learning should take place. The value of the learning rate should be set between 0 and 1; setting a value of 0 indicates that learning does not take place. The initial value is set to

0.9, indicating that learning will take place very quickly, but this does not necessarily mean that learning will take place accurately. In addition, the having a high learning rate shows that the agent values the information gained in each action, which could mean that the agent may find it harder to converge [5].

The discount factor, γ looks at how the agent reacts to the rewards and is set between 0 and 1. The closer the value to 1, the agent looks at maximising the reward structure in the long term, whereas if the value is closer to 0 the agent looks at maximising the reward to its immediate return. The initial value was set closer to 1 at 0.8, to ensure that the agent looks at maximising the long-term reward.

Epsilon ε, the decay factor λ and policy πhave all been defined above.

Updating Q-Matrix

We show how the Q-Matrix will be updated per learning episode. We have set the restriction that the agent can only start at state 0 and not at a random state. The agent has no prior knowledge of how to solve this environment, hence the initial Q Matrix is empty. We consider an episode to be complete once the agent reaches state 33 (goal state). The equation below shows how the Q matrix is updated.

$$Q_{new}(s,a) = Q_{old}(s,a) + \alpha[(r(s,a) + \gamma \max Q(next.s, all.a)) - Q_{old}(s,a)]$$

The process of updating the Q-matrix is as follows for one episode:

1. Create empty Q Matrix Figure 7
2. Initialise the starting state $(s_0) = 0$
3. Define the possible actions from s_0 – 0->1 or 0->6
4. Choose action a_0 according to policy π (epsilon greedy policy)
    a. As both actions give the same reward of 0 the agent will randomly select the next transition state
    b. A greedy action has been chosen hence the next transition state $s_1$will be 1
5. Q matrix will be updated at this point for (s_o, a_0) at t_1
    a. Q value update : $Q(0,1) = Q(0,1) + \alpha(r(0,1) + \gamma * \max(Q[1,:]) - Q(0,1)$
    b. Q matrix update: $Q(0,1) = 0 + 0.9 * (\ 0 + 0.8 * 0 - 0\ ) = 0$
6. New state s_1 after transitioning from s_0 is 1
7. Define the possible actions from s_1 – 1 - >0 or 1 -> 2 or 1-> 7
8. Choose action $a_1$according to policy π (epsilon greedy policy)
    a. All three possible state transitions give the same reward of 0, the agent will randomly select the next transition state
    b. A greedy action has been chosen hence the next transition state $s_2$will be 2
9. Q matrix will be updated at this point for (s_1, a_1) at t_2
    a. Q value update : $Q(1,2) = Q(1,2) + \alpha(r(1,2) + \gamma * \max(Q[2,:]) - Q(1,2)$
    b. Q matrix update: $Q(1,2) = 0 + 0.9 * (\ 0 + 0.8 * 0 - 0\ ) = 0$
10. New state s_2 after transitioning from s_1 is 2
11. Iterations continue until the target state 33 is reached
12. The final update of the Q matrix in the first episode is as follows:
    a. Agent is in state 32
    b. Define the possible actions from state 32 – 32 - > 31 or 32 -> 33
    c. A reward of 100 is given to the agent for transitioning to state 33 (final state)
    d. Q value update : $Q(32,33) = Q(32,33) + \alpha(r(32,33) + \gamma * \max(Q[33,:]) - Q(32,33)$
    e. Q matrix update: $Q(32,33) = 90 + 0.9 * (\ 100 + 0.8 * 0 - 90\ ) = 90$
13. Target state 33 is reached, end of the first episode. Final Q 0 Matrix for first episode can be seen in Figure 8.

Initial Q Matrix	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7 Initialised Q - Matrix

Deep Learning Coursework: Optimization

Vithurshan Vijayachandran and Hisho Rajanathan

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	-90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-90
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	90	0	0
35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8 Q-matrix after one episode

Performance vs Episodes

Initially, a random agent was run through the environment to be used as a comparison. This is to identify the Q agent's learning of the environment over time as compared to that of the random agent. Figure 9 shows how the reward varies over the number of episodes for the random agent. The random agent moves by executing random actions across the environment [4]. We can see that the random agent does not learn, even past 100 episodes, although there are occasions where the agent does achieve the maximum reward of 100. Besides that, at the 80th episode, the agent is seen to achieve a reward of -1000. Figure 9 also shows the cumulative reward of the random agent, which reiterates that the random agent does not learn as the cumulative reward shows that the agent consistently obtains negative rewards over 100 episodes.

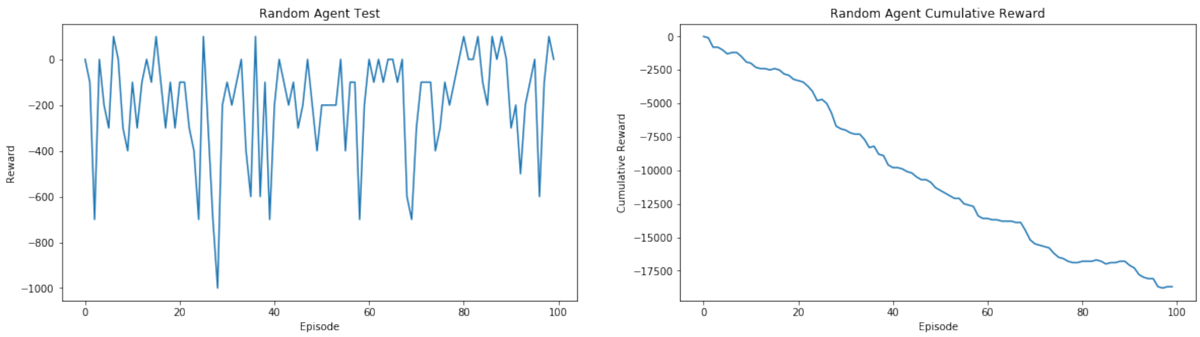

Figure 9 Random Agent

Using the initial parameters highlighted in Figure 6, we can see that the Q-Agent has learnt to try and achieve the maximum reward in the environment. The Q-Learning algorithm was run over 100 episodes, giving a chance for the Q-learning agent to learn the environment as shown in Figure 10. After 10 episodes, the Q-Learning agent has managed to achieve the maximum reward of 100. For most of the episodes learnt, the agent achieved the maximum or an overall reward of 0. Comparing the random agent to the q learning agent, it can be seen that the Q learning agent was able to learn the environment and achieve the maximum reward sufficiently early. This is demonstrated in the cumulative reward graph, where there is a close linear relationship between the cumulative reward and number of episodes. After 10 episodes, the agent achieves more negative returns. However, the agent continues to learn to eventually obtain the maximum return.

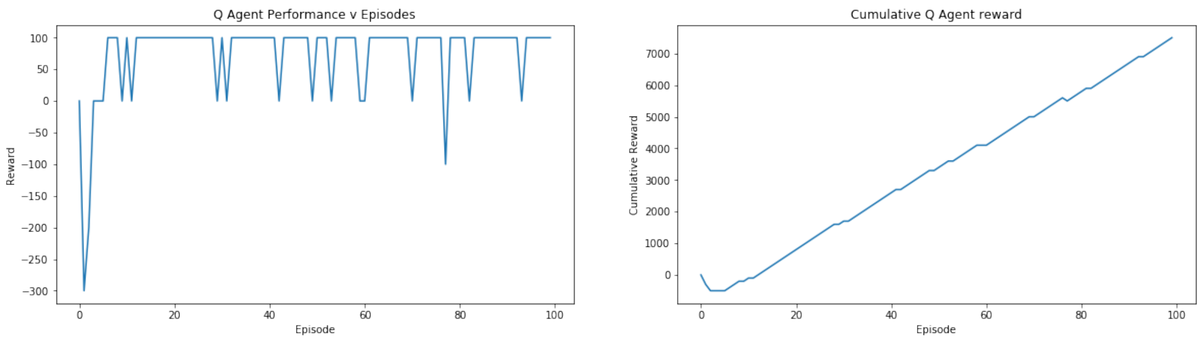

Figure 10 Q - Agent Reward per episode

Deep Learning Coursework: Optimization

Vithurshan Vijayachandran and Hisho Rajanathan

Changing parameter values – Analysis

The different parameters in the Q-Learning algorithm were varied in order to see how the reward changes per episode. Below, the results of varying the learning rate, discount factor, epsilon and decay factor is shown.

Learning Rate, α

As stated above, the learning rate determines how fast learning takes place. However, a higher alpha value does not necessarily mean that learning takes place more accurately. Figure 11 shows how varying the learning rates, changes the rewards per episode. An alpha = 0.001 is observed to give the best results; from the cumulative reward graph, this learning rate enables the agent to receive the highest cumulative reward over 100 episodes. On the other hand, an alpha = 0.1 is observed to enable the agent to receive higher reward initially. However, with the increase in the number of episodes, the agent does not learn sufficiently. Thus, setting the learning rate at alpha = 0.01 indicates that, although learning takes longer, the results will be more accurate.

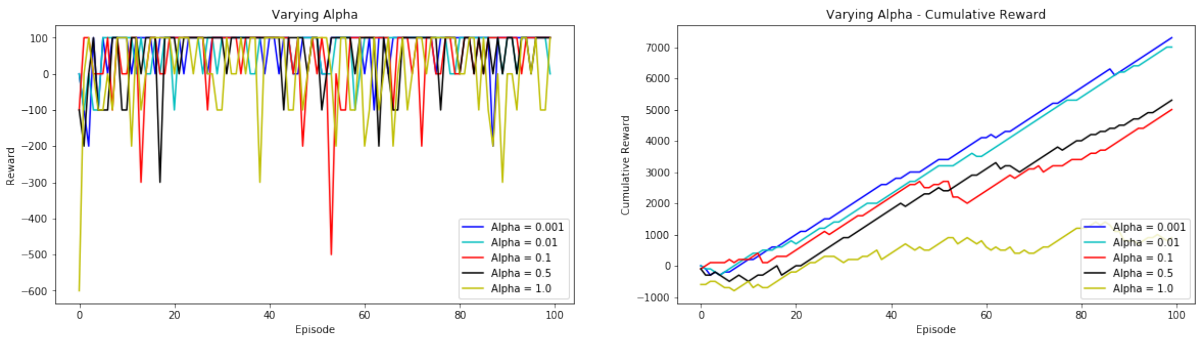

Figure 11 Varying Learning Rate

Discount Factor, γ

The discount factor indicates the importance of future rewards. While the initial discount factor was set at 0.8, this experiment included the study of different discount factors, such as 0.01, 0.1, 0.3, 0.8 and 1.0. This is to study the effects of the discount factor on the change in rewards per episode. A lower discount factor causes the agent to consider short term rewards and a higher discount factor causes the agent to consider long term rewards. From the graph above, it can be seen that a lower discount factor of 0.01 enables the agent to perform better for each reward per episode. The graph converges faster when compared to the graphs of other discount factors and consistently achieves a near-maximum reward per episode. The discount factor of 0.01 is able to achieve a much higher reward over the 100 episodes, indicated by the steeper gradient in the cumulative rewards graph, compared to the other discount factors.

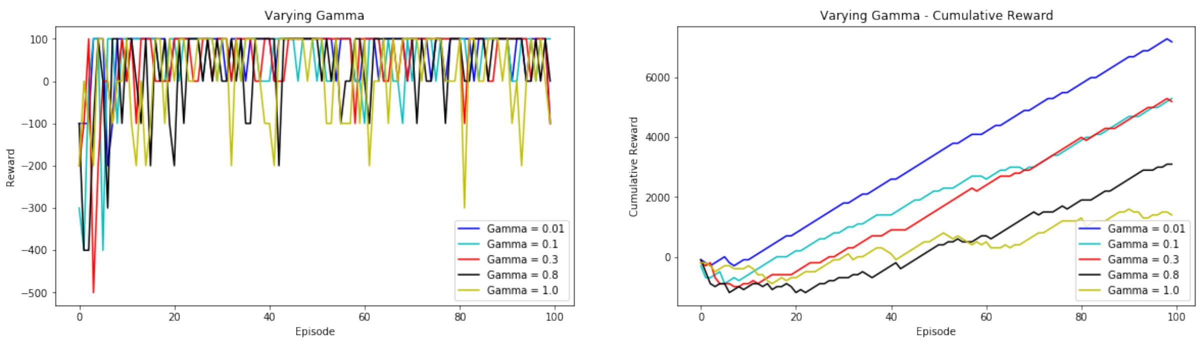

Figure 12 Varying Discount Factor

Epsilon, ε

Similar to the previous hyperparameters, the epsilon value is varied between 0.1 and 1.0, where the initial epsilon value was set at 0.9. From Figure 13, setting the epsilon value at 0.75 proves to achieve better rewards per episode. The agent learns to achieve the maximum reward sufficiently quick and the reward does not fall dramatically, as compared to the other epsilon values. Setting the epsilon high, as fixed in the initial model, does not prove to be efficient as the agent consistently achieves the lowest reward possible. All other possible epsilon rates indicate that the agent achieves negative rewards, apart from an epsilon rate of 0.75. Setting the epsilon rate at 1 indicates that the agent focuses on exploring. This causes the agent to only achieve negative rewards.

However, setting the epsilon rate at 0.75 allows the agent to exploit the environment, enabling it to achieve better rewards.

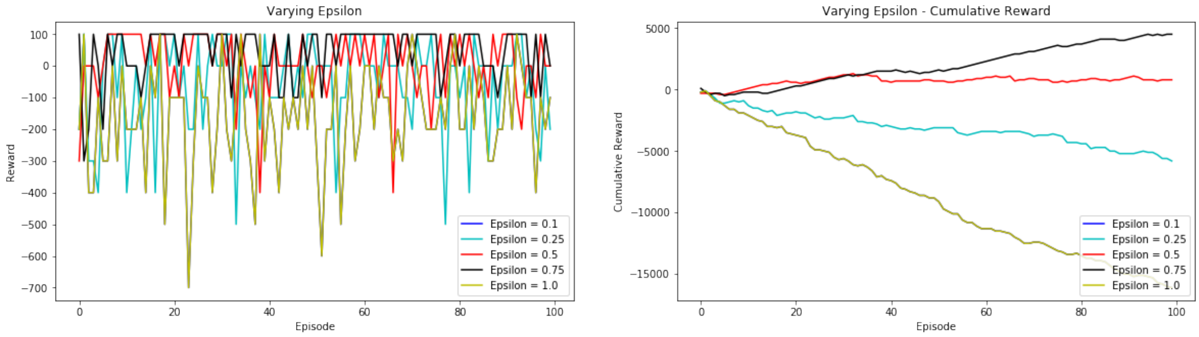

Figure 13 Varying Epsilon value

Decay Factor,λ

The decay factor was applied to the epsilon greedy policy, which allowed the agent to start by exploring the environment and eventually exploiting it. Figure 14 shows how varying the decay factor changes the reward per episode. The higher the decay factor, of either 0.9998 or 0.998, the better the results as the agent is seen to not have rewards below -100. While There is not a significant difference between a decay factor of 0.9998 or 0.998, a decay factor 0.9998 achieves higher rewards marginally at the later episodes. The lower the decay factor, the less the agent learns with time (number of episodes). This is clearly observed as the rewards obtained from an agent gets worse.

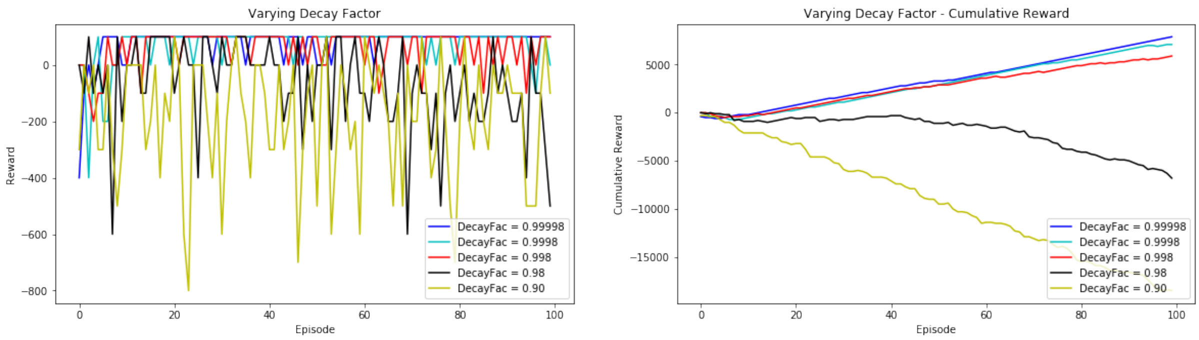

Figure 14 Varying Decay Factor

Conclusion

Implementing Q – Learning to the grid problem outlined in Figure 1 was easy and effective, as the agent was able to learn the environment sufficiently quick within 100 environments. Comparing the initial random agent to the Q learning agent that was deployed into the environment, it can be seen that the random agent did not learn the environment by taking random actions. The Q-Learning agent was able to learn the environment by avoiding the states with the negative rewards as shown in Figure 10. Keeping in mind the broader arch of this study, future work can involve the change the policy π to such as using the Boltzmann Policy, as opposed to the Epsilon Greedy Policy.

As the environment implemented in this study is a simple environment, it was not necessary to combine the use of reinforcement learning with neural networks such as Deep Q Network or Double Deep Q Network. To make the grid move complex, the board could be extended to environment of a 10x10 grid and have intermediate rewards that the agent could achieve before receiving the final state. This could potentially help in improving the time spent for the agent to learn how to maximise the reward. Another method that could also be beneficial is to allow the agent to start at any random state in the board and navigate its way the final reward state.

We did look into using absorbing states for all the negative reward states, however the Q – Learning agent did not learn the environment as quickly and we would have had to use a higher number of episodes.

Deep Learning Coursework: Optimization

Vithurshan Vijayachandran and Hisho Rajanathan

Advanced

This section will study the implementation of deep reinforcement methods to the MountainCar environment on Open AI Gym[1]. The aim is to navigate the vehicle in between the two mountains in one single pass - the only possible way of doing this is by the vehicle building up momentum by driving back and forth, due to the vehicle's engine not being powerful enough.

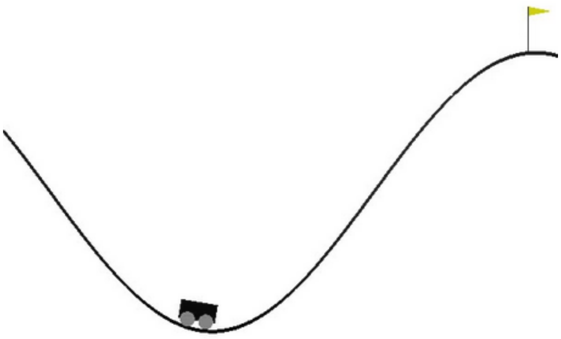

Figure 15 Mountain Car Environment from Open AI Gym

In the case of the Mountain Car problem, the agent is the car and the reward is given to the car once the agent receives the yellow flag. As part of the requirements of this environment, the game will finish once 200 episodes are completed. The car has three possible actions in this environment: accelerate to the left, do not accelerate and accelerate to the right [6]. The agent is given a reward of -1 once it reaches the yellow flag at the top of the right mountain.

We will be looking at implementing Deep Q Network (DQN) and Double Deep Q Network (DDQN) to try and solve this environment.

Deep Q Network (DQN)

Deep Q Network (DQN) is built based on the Q-Learning algorithm and is the first deep learning technique released by DeepMind for playing the Atari game [7]. The Q-Learning algorithm is seen as inefficient as the results are stored in a tabular form (in a Q table), causing an increase in computational cost, and is limited to the specifications of the machine used to run the algorithm on. The introduction of Deep Q – Learning solves this limitation by using neural networks to store the values of the Q – Learning algorithms, where additional layers can be added to the neural network. However, the DQNs are not preferred due to its unstable nature as it suffers from some overestimation in certain games in the Atari network [8].

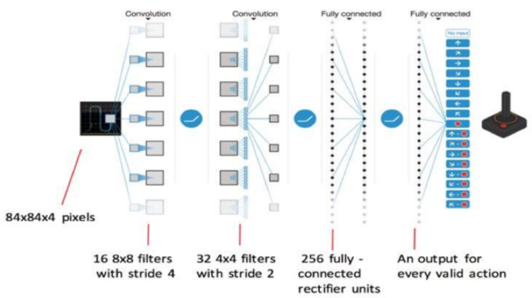

Figure 17 Example of DQN in an Atari Environment

$$Q(s, a; \theta) \approx Q^*(s, a)$$

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a\right]$$

Figure 16 Evaluation of DQN to approximate Q

Double Deep Q Network (DDQN)

The Double Deep Q Network was introduced to combat the overestimation caused by the Deep Q Network. The agent in a DDQN uses two neural networks, compared to one in DQN, to learn the environment and predict which action to take at each timestamp, depending on the reward [9]. In DDQN, the agent learns and improves itself using a process called experience replay, used in the second neural network in this specific architecture.

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^-)$$

Figure 18 Evaluation of Double Deep Q Learning Network [8]

Methodology

Algorithm

The following algorithm was used to solve the Mountain car challenge using DQN. Modification were made suit the DDQN and DDQN with PER networks.

Initialise Experience Memory

Initialise Q Network

For episode = 0, MAX_EPISODES

[1] https://gym.openai.com/envs/MountainCar-v0/

Initialise environment

For step = 0, MAX_STEPS

Action = ε – greedy

Execute action

Observe next state and reward

Record the learning cycle

Train model

End

Reward Structure

Keeping to the original reward structure, it is observed that the agent took a sufficiently long to achieve the reward as the environment terminated at 200 episodes. During this time, there were many instances where the car did not reach any point near the yellow flag, situated at the top of the mountain. As a result of this, it was decided that the reward function be altered to allow the agent to learn the environment in a quicker manner. The cart receives the following awards for crossing each line shown in Figure 20; 10 for crossing the blue line, 40 for the crossing yellow line, 90 for crossing the green line and 190 for crossing the red line.

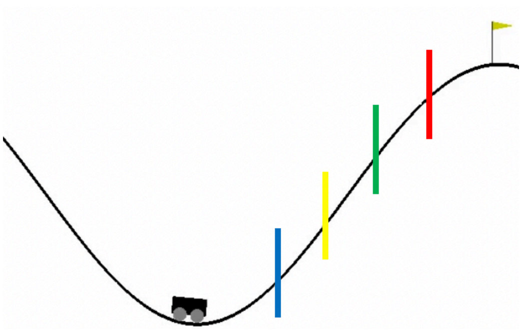

Figure 19 Reward structure

Parameters

Parameter	Value
Alpha	0.0001
Gamma	0.97
Epsilon Start	1.0
Epsilon End	0.01
Policy	Epsilon-Greedy
Decay factor	10000
Hidden layers	2
Hidden Neurons	50
Activation Function	ReLu
Optimizer	Adam
Batch Size	16
Max Memory	50000

Figure 20 Parameter values

Due to the computational constraints, each network was trained over 1000 epochs. This was decided as, when opted for a higher number of epochs such as 10,000 epochs or 5000 epochs, the machines continuously crashed. Therefore, we decided on a smaller epoch size of 1000. In order to make ensure a fair comparison, a constant set of parameter values were used to train all the networks. Chosen parameters are shown in Figure 19.

Analysis

Training

DQN

During training, the performances of the models were measured for evaluation. As shown in Figure 21, the average rewards attained by the agent increased with the increasing number of episodes, indicating that the agent is learning to reach the target. The average rewards obtained by the agent showed an initial unstable pattern. However, at approximately 600th episode, the average reward consistently increased. Therefore, it is safe to assume that the agent had successfully learned to reach the target around the 650th episode as the average reward had a positive upward gradient. Upon further inspection of the episode length, it is evident that the agent may have learned to reach the target after approximately the 150th episode, as the number of steps taken within an episode had reached a minimum value. This is only possible once the agent has reached the target state. However, from about 900th episode, the number of steps were increasing while the reward achieved remained high on average.

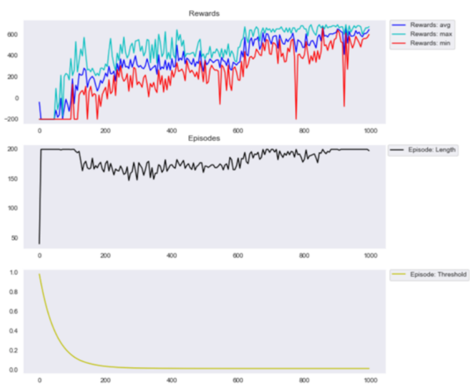

Figure 21 DQN Training Results

Deep Learning Coursework: Optimization

Vithurshan Vijayachandran and Hisho Rajanathan

DDQN

As expected, Double Deep Q Network had a better overall performance than DQN, as shown in FIG 22. Comparing the rewards with DQN, it is clear that DDQN had reached its target at approximately the 600th episode. This is, again, demonstrated by the minimum point of average rewards before a continuous positive gradient. Inspecting the episode length, the DDQN model saw a drop-in number of steps from about 180th episode till the 400th episode and from 400th episode, DDQN requires the maximum number of steps. From 400th episode, the stable patter of max reward showed the agent constantly achieving high reward. However, for the same period, the minimum award shows that the agent was constantly exploring to achieve a higher reward.

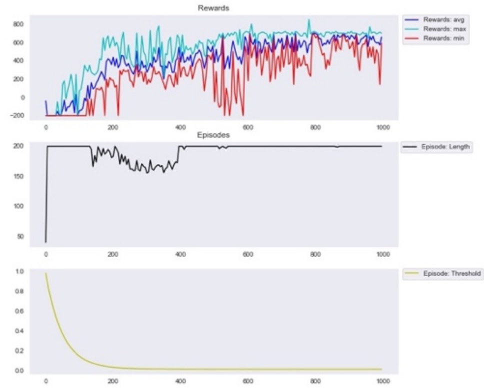

Figure 22 DDQN Training Results

DDQN with PER

To further study the extent of DDQN, Prioritized Experience Replay (PER) [10] was implemented together with DDQN. PER prioritizes and replays experiences where the agent learnt the most, as opposed to selecting experiences from the buffer randomly. This can be achieved by calculating the temporal difference error during the time of training and using a probability distribution to sample them.

From the results obtained, it is surprising to see that compared with DQN and DDQN, the network only started to gain rewards from about 200th episode and average, minimum and maximum reward remained below 0. From that point onwards, the model's average reward was on an uphill rise. Inspecting the episodic length, the model had obtained a maximum number of steps till the 200th episode. From there on the number steps were reduced indicating the convergence of the model. However, from about 400th episode, the model takes maximum number of possible steps. This could be due to the maximum reward achieved around 250th episode which was never repeated. Indicating further training is required.

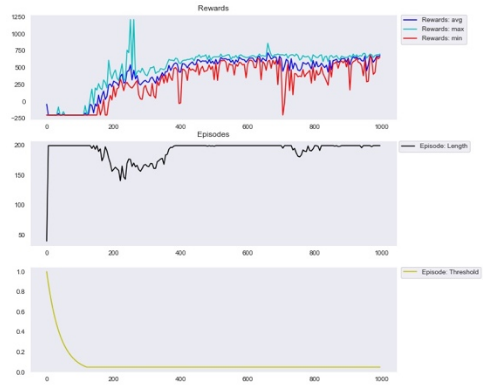

Figure 23 DDQN with PER Results

It is evident that the three networks didn't manage to fully converge over 1000 episodes as the number of steps were utilised to the maximum. However, it is also worth remembering that we modified our reward structure. This in turn have triggered the agent to learn to stay beyond the red line to achieve maximum number of reward which will in turn require maximum number steps to be utilised.

Test

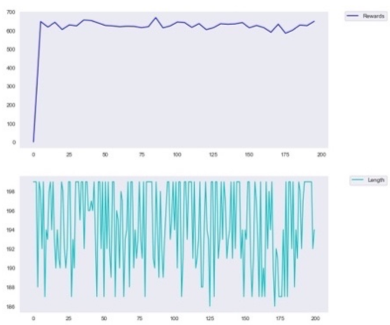

Figure 26 DQN Results

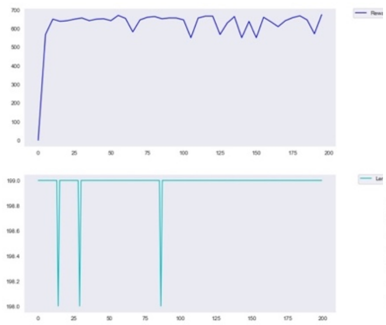

Figure 25 DDQN Results

Figure 24 DDQN with PER Results

From the visualised results shown in Figure 26, 25 and 24, it is evident that DDQN with PER turned out the the best performing model as expected. Results for DQN showcase that the model require further training as the number of steps varied per episode indicating the model didn't fully converge during training. DDQN however, was more stable with the number of steps over episodes, but the reward achieved indicates the it also requires further training. DDQN with PER on the other hand, remained constant with number of steps over episode and reward achieved over 100 episodes were constant indicating the agent successfully learned to reach the target and stay beyond the red line to achieve maximum number of rewards while utilising maximum number of steps.

Conclusion

The DQN is comprised of a neural network of multiple layers, on the other hand, the DDQN is comprised of two neural networks, which uses experience replay to improve the agent's self. For a more complex environment, the both methods proved to be beneficial in its performance. In the MountainCar environment, a car was required to move between two mountains using momentum. Different valued rewards were given at different points on the mountain. It was proven that throughout the training stages, the DQN agent was able to learn in a quick manner, although trained on a smaller number of epochs of 1000. The agent was seen to reach the target at the 600^{th} episode, with a consistently high reward attained. However, the DDQN agent had outperformed the DQN agent, as it was seen to reach the target at the 500^{th} episode.

An extension of DDQN -DDQN PER- was also studied. This model, which places an emphasis on experiences that benefited the agent's learning the most, had shown better performance with than DQN and DDQN. It was able to reach its target at the 200^{th} episode. Despite the slight deviation in the number of steps of an episode later, at episode 750, this model had an overall more stable performance when compared to the previous models. Hence, it is the most reliable method.

From our study it is evident that deep reinforcement learning requires sufficiently large computational powers. Due to the lack of access to computers with GPU's, we were unable to find the optimal hyperparameters and train the networks over larger episodes (50,000 rather than 5000).

References

[1] Wiering, M. & van Otterlo (Eds.), M., 2012. *Reinforcement Learning.* s.l.:Springer.

[2] Zychlinski, S., 2019. *The Complete Reinforcement Learning Dictionary.* [Online]
Available at: https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e

[3] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA.

[4] Novatec. 2018. *Reinforcement Learning – Part 1: Introduction To Q-Learning | Novatec*. [online] Available at: <https://www.novatec-gmbh.de/en/blog/introduction-to-q-learning/>.

[5] Medium. 2019. *Reinforcement Learning From Scratch: Simple Application And Evaluating Parameters In Detail*. [online] Available at: <https://towardsdatascience.com/reinforcement-learning-from-scratch-simple-application-and-evaluating-parameters-in-detail-2dcee3de008c>.

[6] GitHub. n.d. *Openai/Gym*. [online] Available at: <https://github.com/openai/gym/blob/master/gym/envs/classic_control/mountain_car.py>.

[7] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan and Riedmiller, Martin. "Playing Atari with Deep Reinforcement Learning." (2013):

[8] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double Q-Learning. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16). AAAI Press, 2094–2100.

[9] mc.ai. 2020. *Introduction To Double Deep Q Learning (DDQN)*. [online] Available at: <https://mc.ai/introduction-to-double-deep-q-learning-ddqn/>.

[10] Schaul, T., Quan, J., Antonoglou, I. And Silver, D. 2015. "Prioritized Experience Replay" arXiv preprint arXiv:1511.05952

Deep Learning Coursework: Optimization

Vithurshan Vijayachandran and Hisho Rajanathan

Contribution

Working with Vithurshan on the deep learning coursework went smoothly especially during the COVID-19 pandemic. We were able to work through the coursework together at each section and provide support to each other. We contributed equally on all parts of the coursework.

It was a pleasure working with Vithurshan and I would happily work with him again.