INM707 Deep Reinforcement Learning Coursework

1.1 Basic Task: Define the Domain, Task and Motivation

In the precursory decades, much progress has been achieved in the field of solving challenging problems across various domains using deep reinforcement learning (*Henderson, P., Islam (2018)).* One of the most successful examples under the spotlight are automated driving, robotics, and decision-making games. Advances in deep reinforcement learning have allowed autonomous agents to perform well on Atari games, often outperforming humans, using only raw pixels to make their decisions (*LAMPLE, G.; CHAPLOT, D. 2017*).

Reinforcement learning works in the following steps. The agent observes an input state, and an action is decided upon by the behaviour of previous trial and error interactions. The action is executed, and the agent will receive a reward. This information about the reword and the corresponding state is recorded and will be used to map out an effective strategy that will achieve the highest rewards.

The environment that this project will focus upon is a classic example of a dungeon where a player must navigate through a static, two-dimensional 5x5 grid World from the start (A) to Finish (Y) whilst avoiding traps in spaces highlighted by red in figure 1. The agent must navigate through the maze and avoid traps which will terminate the session and reward an extreme low negative score. Each step is rewarded as -1 and reaching the goal awards 100. The objective is to reach the end destination in the shortest number of moves possible.

The optimal policy consists of finding the high rewards, avoiding low rewards (traps) and prevent needless navigation to reach the target in least number of steps.

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O
P	Q	R	S	T
U	V	W	X	Y

Figure 1: A and Y are the start and end states respectively and the trap positions are represented by the red tiles.

1.2 Define the State, Transition and Reward Functions

1.2.1 State transition function

A state in this case is simply an individual cell in the grid world identified by a unique character and the actions are the available corresponding states the agent can navigate to. The Start State is always A with the objective to reach Y. The session terminates when it reaches Y or falls into the red states highlighted by figure 1. The session has 6 terminal states.

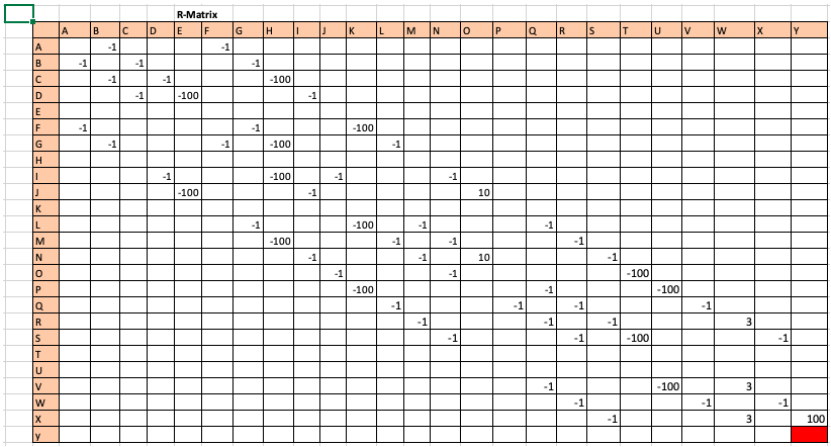
The action space consists of four movements the agent can perform; action = {up,down,left,right].

	UP	DOWN	RIGHT	LEFT
A	A	F	B	A
B	B	G	C	A
C	C	H	D	B
D	D	I	E	C
E	E	J	E	D
F	A	K	G	F
G	B	L	H	F
H	C	M	I	G
I	D	N	J	H
J	E	O	J	I
K	F	P	L	K
L	G	Q	M	K
M	H	R	N	L
N	I	S	O	M
O	J	T	O	N
P	K	U	P	Q
Q	L	V	R	P
R	M	W	S	Q
S	N	X	T	R
T	O	Y	T	S
U	P	U	V	U
V	Q	V	W	U
W	R	W	X	V
X	S	X	Y	W
Y	T	Y	Y	X

Figure 2: transition matrix

1.2.2 Reward function

At each transition, the agent receives the reward moving to the next state defined by the R-matrix in figure 3. A normal transition from state-to-state rewards a score of -1 as the intention is to reach the end node in fewest steps. The agent reaching the end receives 100 which is a positive reward which incentivises the agent to reach this node. Falling into a trap termination note gives the agent -100 points therefore the agent should theoretically avoid these.



R-Matrix

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
A		-1				-1																			
B	-1		-1				-1																		
C		-1		-1				-100																	
D			-1		-100				-1																
E																									
F	-1						-1				-100														
G		-1				-1		-100				-1													
H																									
I				-1				-100		-1				-1											
J					-100				-1						10										
K																									
L							-1				-100		-1				-1								
M								-100				-1		-1				-1							
N									-1				-1		10				-1						
O										-1				-1						-100					
P											-100						-1				-100				
Q												-1				-1		-1				-1			
R													-1				-1		-1				3		
S														-1				-1		-100				-1	
T																									
U																									
V																	-1				-100		3		
W																		-1				-1		-1	
X																			-1				3		100
Y																									

Figure 3: r-matrix

1.3 Set up the Q-learning parameters (gamma, alpha) and policy

For the agent to find an optimal solution, it is imperative that we set a policy which maximises reward. A policy is a mapping from states to probabilities of selecting a certain action at a state. The policy π is defined as the probability distribution over all actions A over every possible state S.

$\Pi(a|s) = P(A = a| S= s)$. where the lowercase represents singular states.

For the project, two different policies were tested: the Boltzmann policy and an Epsilon greedy decay policy.

Boltzmann policy

In this policy an action is chosen based upon weighted probabilities over all possible actions. In this case it is an array of 4 probabilities. The Boltzmann policy normalizes the final Q values using a softmax function and uses the resulting values as probabilities, selecting an action much like a stochastic policy (*Doughlas 2020*). This creates estimation values over each action and the best estimated action is more likely to be selected by a corresponding probability value.

Epsilon Greedy Decay Policy

An Epsilon Greedy Decay algorithm is unique in the sense that it encourages early exploration stages which greatly reduces bias from the early distribution of states. It encourages randomness in selection in the early stages over the higher rewarded states and over time the epsilon value will decay from 1 linearly for a more refinement stage learning from previous iterations.

Parameters for Q learning

The values for the parameters were carefully chosen from manual hyper tuning, the information for which is recorded in figure 6. The following parameters were set:

Alpha: This is the value for the learning rate with range (0,1). This is the amount the weights are updated by in training. A small learning rate will cause a program to run more slowly by requiring more iteration steps to find the optimal solution whilst a large learning rate will diverge much too quickly which trades accuracy and the model will not be precise. After tests, learning rate was set to 0.3.

Gamma: This is the discount factor which signifies how important we place the rewards for future functions. This ranges from 0 to 1. A value closer to 0 will put more consideration to immediate rewards whilst a value closer to 1 looks at more long term. In this case it was set to 0.99 which had the best overall rewards.

Episodes and steps: An episode is a single run of All states that come in between an initial-state and a terminal-state and a step is simply the number of max steps to terminal state before terminating.

T: This is unique for the Boltzmann policy where T is a constant that affects action vector which controls the amount of exploration. This was set to 0.5

Epsilon: Unique to epsilon decay, this is set to 1 to allow greatest exploration.

Epsilon_min: Lowest value Epsilon can be after decaying.

Decay_rate: Linearly how the Epsilon value falls by.

1.4 Run the Q-learning algorithm and represent its performance

Example of Learning Episode

1. Initialise an empty Q table of 25x4 and fill with 0.
2. Agent Selects an action based on its policy.
3. State moves from State 1 to State 2 from the corresponding action
4. Agent will receive the reward
5. Update the Q table by Q-learning update rule by following formula

Q(state, action) = Q(state,action) + alpha*(reward + gamma* max(Q(new state) - Q(state, action))

Below is an example of iteration

```
Action Probability Distribution:  deque([3.594315415931898e-103, 1.0, 2.3987432834376526e-86, 8.86981543341676e-87], maxlen=4)
Action DOWN
Current state M
Next state R
Done False
Old Q value:        97.943535
Updated Q value (Q new):  97.94967105102539
test -0.010000000000000009
Action Vector from Q Table UP       -0.700719
DOWN     -0.895372
RIGHT    98.984627
LEFT     -0.828980
Name: R, dtype: float32
Action Nominator:  deque([0.24624272759534957, 0.16683604819104017, 9.481926102404226e+85, 0.19052746110404278], maxlen=4)
Sum Denominator 9.481926102404226e+85
SIZE Action Nominator:  4
```

Performance of Model

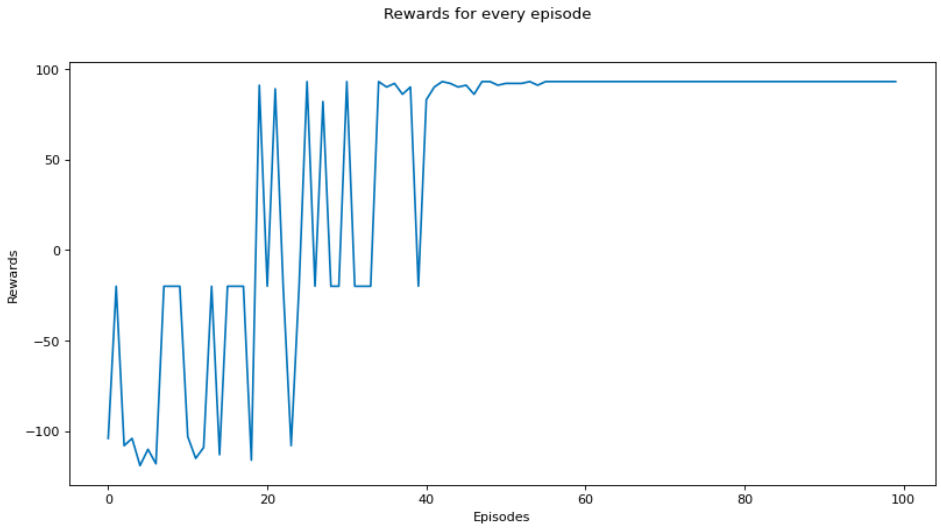


Figure 4: Boltzmann policy implementation rewards vs episodes

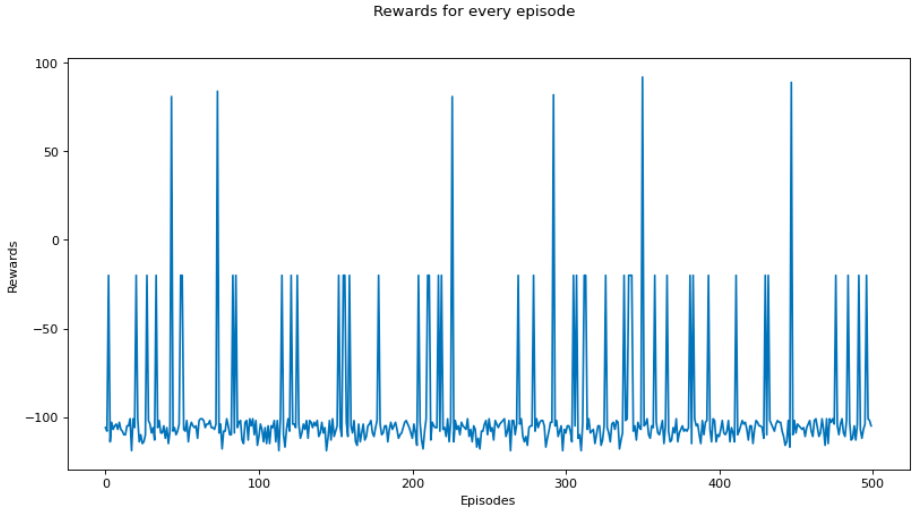


Figure 5: Epsilon Decay policy implementation rewards vs episodes

1.5Repeat the experiment with different parameter values, and policies

From Figures 4 and 5 earlier it is clear that the Boltzmann method is superior to the Epsilon Decay algorithm. The Boltzmann quickly converges to the maximum reward possible with 56 steps while Epsilon policy is much too slow, increasing decay rate and learning rate appear to make no significant changes to the model.

With Boltzmann Q policy there is a clear effect on parameter hyper tuning and the number of episodes to reach max reward.

Gamma Alpha	0.5	0.7	0.9	0.99
0.1	103	95	84	58
0.2	102	67	62	56
0.3	57	51	46	37
0.5	71	59	49	43

1.6 Analysis

This was a very successful project as the agent manages to navigate through the grid with minimal steps especially from the Boltzmann policies. The only disadvantage of the Boltzmann policies is the time to run multiple episodes which is my this is limited to 100 in the code. It is the belief that the epsilon decay would have been successful, but it requires much greater number of episodes than 500 which was the maximum amount that was psychically possible by the computer.

The Boltzmann regularly outperformed the decay algorithm and it requires fewer episodes therefore it is the better model and the code also considers manual hyper tuning where of 16 different alpha and gamma values the best were selected. Gamma appears to play a more significant role in limiting the number of episodes than alpha.

Boltzmann model is far more methodical and complex where it doesn't need an exploration period and the weighted probabilities assigned to each action makes a massive difference in reducing episode length as the agent realises there are actions such as staying in the same state or moving further away which offer less reward therefore is not worth it.

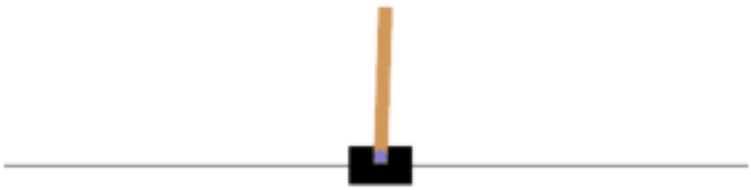
It would have been interesting to compare this to other methods such as greedy epsilon and perhaps consider a more complex larger grid with different variables including multiple rewards or different negative reward systems to make the state more complex which should give a more accurate analysis of these policies.

2. Advanced task

Domain and Task

Automating games to achieve high scores has become something of a craze in the past few years; programmers are building programs to push scores beyond human capabilities. A team of researchers at Uber AI Labs in San Francisco has developed a set of learning algorithms that proved to be better at playing classic video games than human players through experiments (*Bob Yirka 2021).* Creating such models can be complex and time-consuming.

This project will be employing OpenAi gym to simulate an Atari CartPole game. The objective is to balance the pole which is connected to a moving slab for as long as possible. There are two actions in this scenario which is to move left and right. The environment will be making observations such as pole and cart velocity, pole angle and cart position. The reward is 1 for every step taken for cartpole, including the termination step.



Define the State, Transition and Reward Functions

The state is represented by the array from environment observation{pole Velocity, cart velocity, pole angle, cart position}

The action space is consisted of 2 actions represent by 0 and 1 which is to accelerate left and right respectively.

A reward of +1 is given for every step the pole remains upright. The episode ends when the pole is more than 15 degrees from the vertical or there is a termination state and reward if the cart moves more than 2.4 units from its initial position.

Implement DQN with two improvements to Atari Learning Environment

The objective of DQN is to combine reinforcement learning and deep learning which can take in large inputs and decide on actions to maximise the game score. It combines uses of neural networks and the framework of reinforcement learning q learning policies to create a complex, high performing solution to problems. There are limitations in the basic model where it can overestimate true rewards and of Q values and they usually have a slow learning rate.

The first objective was to identify which methods can boost the algorithm in achieving the required rewards in an earlier time frame.

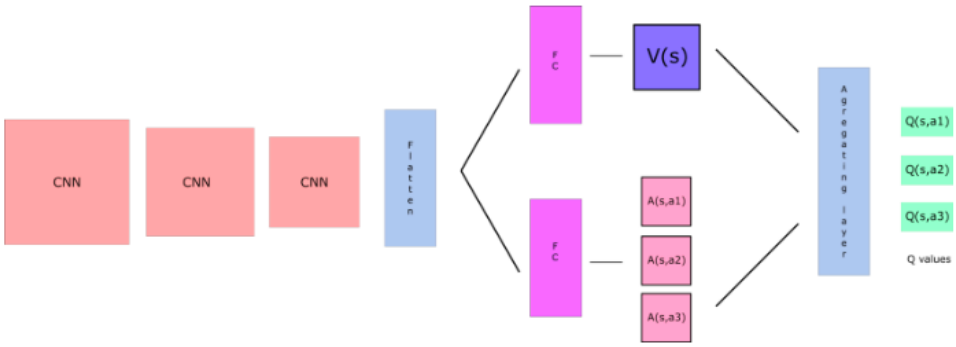
This was achieved using the rllib and tune python modules from ray. This method was inspired from labs 6-9. The program defined an evaluation function which analysed different parameter combinations based on episode means of 30 episodes after training, an objective function, that allows us to set how the rl agent is trained depending on the parameters. A hyperparameter search loop was implemented to consider all parameter combinations and after days of running, a method was chosen involving dueling and double DQN and gamma value of 0.97. A greedy epsilon policy will be used.

Dueling DQN

In this architecture there are two separate streams within the network that estimate different functions, the state value, and the advantage of each action. After the two streams the last module combines the two information and optimises an output. This greatly reduces the number of episodes required to achieve optimal solution. Furthermore this value calculates V(s); This is particularly useful for states where their actions do not affect the environment in a relevant way.

$Q(s,a) = A(s,a) + v(s)$

Where V(s) is value of state and A(s,a) is the advantage of taking action a at state s



Dueling Architecture, Source: https://www.freecodecamp.org/news/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682/

Double DQN

Double DQN utilises double q-learning to reduce overestimation by decomposing the max operation in the target into action selection and action evaluation (*Hasselt et al 2015*). It also consists of two networks which are an online network to select best action and a target network to evaluate the action with different set of parameters.

$Q(s,a) = r(s,a) + \gamma maxQ(s',a)$

Where r(s,a) is the reward of taking that action at that state and $\gamma maxQ(s',a)$ is the discounted max q value from all possible actions from next state.

Analyse the results quantitatively and qualitatively

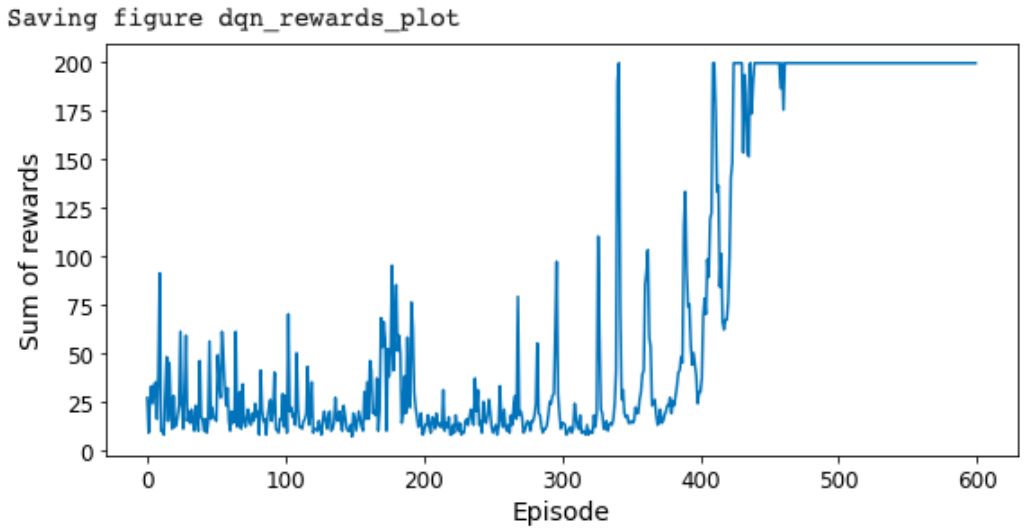


Figure 7: DDDQN model on rewards vs Epsiode length

The optimal model was already predetermined from ray tuning and running a simple DQN took around 484 episodes therefore there is a significant progress in creating the best model. The initial optimisation script took days to run and cartpole is considered one of the simplest games out there with only two actions and limited space so this method may not be ideal for more complicated games with more actions and states available and you would have to limit the range of parameters in the hyper tuning step.

This methodology was very much inspired by the labs 7-9 exercises where different configurations of parameters would be run, and this is analysed by an accuracy score which outputs the best parameter combinations. The only downside is the length of time.

The greedy epsilon policy was implemented for simplicity of the model where if random value is greater than epsilon then the preferred outcome is selected else the action is chosen uniformly between all actions. This was a simple yet effective method which isn't as time consuming as epsilon decay and others as fewer steps are implemented but in a future exercise different policies should definitely be considered. The replay buffer is extremely important in this case as initial samples are randomly selected and the refinement is applied afterwards after brief period of exploration. This is visible in the graph above where up to 400 episodes its almost cyclic and random and afterwards it converges on the maximum reward.

This projected was completed individually by Hrushik Tadvai

References

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep Reinforcement Learning That Matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

LAMPLE, G.; CHAPLOT, D.. Playing FPS Games with Deep Reinforcement Learning. AAAI Conference on Artificial Intelligence, North America, feb. 2017

Douglas De Rizzo Meneghetti Visualizing temperature in a Boltzmann policy
May 2020

Reinforcement learning algorithms score higher than humans, other AI systems at classic video games Bob yirka, may 2021

Deep Reinforcement Learning with Double Q-learning Hado van Hasselt, Arthur Guez, David Silver

September 2015