Deep Learning Optimization - Coursework

Author: Abin Abraham

1. Introduction

The goal of reinforcement learning is for an agent to learn an optimal policy that maximizes its cumulative future reward[1]. One broad method of classification is based on the approach to how the agent learns - value based or policy based. The former involves trying to learn the value of each action, given the state and indirectly learns the policy by performing actions that maximize value, while the latter attempts to learn the policy directly. Value measures the worth of the agent's action beyond the immediate reward. Q-learning is a value-based approach which learns by trying to find each state's action-value function. To choose which action to take for given a state, we take the action with the highest Q-value (maximum expected future reward we will get at each state). Q-learning only works in environments with discrete and finite state and action spaces. A solution for extending Q-learning to complex environments is to apply neural networks to learn the value function, taking states as inputs, instead of storing the full state-action tables as in q learning. Deep Q networks are a collection of such algorithms. They use neural networks as function approximators to approximate the value function. To ensure convergence, it also includes improvements like experience replay, target network, duelling etc. The list of algorithms includes – Deep Q learning with experience replay and fixed target, Double Deep Q Learning, DQN with Prioritized Experience Replay and Duelling DQN.

Through this coursework, we will work with value-based reinforcement learning algorithms which are model free i.e. the agent does not have access to the environment model. It is divided into two parts. The first part implements and analyses Q learning through a task to identify the shortest path in a transport network and the second part implements and analyses two Deep Q learning algorithms through solving the Lunar Lander v2 environment from OpenAI Gym.

2. Basic – Q Learning

2.1 Define a domain and the task

Figure 1: London Tube Map - East

The domain chosen is a virtual transport path modelled on the Tube or London Underground Map which has multiple lines crossing paths and some stations which are accessible via more than one line and some which are accessible only via a single line. We will focus on the eastern section of the tube. The task is to identify the **shortest path** between any station in the map to a destination station with reinforcement learning using the Q-learning algorithm. In Q-learning, the expected total long-term reward given a state 's' and action 'a' is predicted by the Q-function Q(s,a). Q(s, a) function, gives us an estimate of the discounted sum of rewards of taking action a in state s. The Q values are recursively updated using a modified form of the **Bellman Equation** to learn the optimal policy.

$$Q_{New}(s_t,a_t)=(1\text{-alpha})Q_{Old}(s_t,a_t)+\text{alpha}(r_t+\text{gamma}*\max_{a'}Q(s_{t+1},a_{t+1})) \quad \text{– eqn 1}$$

where alpha is the learning rate, gamma is the discount factor, Q_{Old} is the current Q value for state s_t and action at, Q_{New} is the updated value, $max_{a'}Q(s_{t+1},a_{t+1})$ is the Q value for the next state when taking the action that maximizes its Q value.

2.2 Define a state transition function and the reward function

The state transition function defines the range of states that the agent can proceed to, from its current state. The table below describes the state transition for the 20 states – 0 to 19 where the left node is the current state and the nodes on the right are the possible next states. Note that the agent can also choose to remain in the same state.

0->{4, 3}	5->{1, 2, 6, 7}	10->{8, 9, 16}	15->{14, 16}
1->{2, 4, 8, 9, 5}	6->{2, 5}	11->{4}	16->{15, 10}
2->{1, 3, 6, 5}	7->{5, 17}	12->{3, 13}	17->{7, 18}
3->{0, 2, 12}	8->{1, 9, 10, 14}	13->{12}	18->{17, 19}
4->{0, 1, 11}	9->{1, 8, 10}	14->{8, 15}	19->{18}

The destination id defined as station 10. The reward function has been defined such that there is a penalty on trying to use a path where there is no connection between two stations. Between two stations without a connection , the reward is -100. Between each station with a connection , the reward is 1. For the penultimate stations connected to the destination and for continuing to remain in in the destination node, the reward is defined as 100.

2.3 Represent the problem with a graph and set the original R-matrix

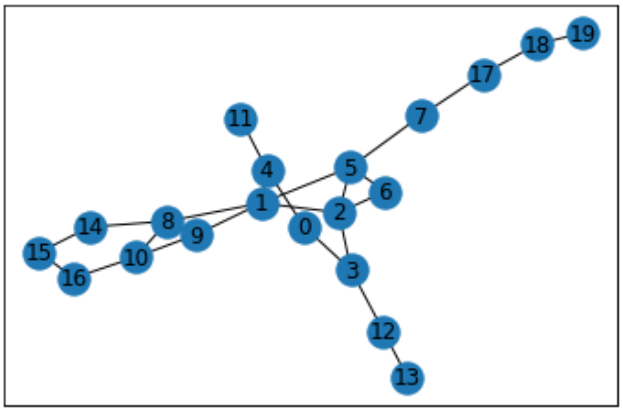


Figure 2: Graph for the network

The virtual map is represented a collection of nodes and edges. Each node represents a station and an edge the connection between any 2 stations. There are many nodes that are only accessible via one route E.g.: 11->4 and are only connected to 2 other nodes E.g.: 12->3,13 There are nodes which are connected to more than 2 other nodes E.g.: 1->2,4,5,8,9

The station numbers displayed in the diagram are arbitrary and have no semantic meaning apart from identification.

R matrix defines the immediate reward for taking an action a in state s and is set as per the reward function in the earlier section. Observe the rewards for destination node 10.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	-100	-100	-100	1	1	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100
1	-100	-100	1	-100	1	1	-100	-100	1	1	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100
2	-100	1	-100	1	-100	1	1	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100
3	1	-100	1	-100	-100	-100	-100	-100	-100	-100	-100	-100	1	-100	-100	-100	-100	-100	-100	-100
4	1	1	-100	-100	-100	-100	-100	-100	-100	-100	-100	1	-100	-100	-100	-100	-100	-100	-100	-100
5	-100	1	1	-100	-100	-100	1	1	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100
6	-100	-100	1	-100	-100	1	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100
7	-100	-100	-100	-100	-100	1	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	1	-100	-100
8	-100	1	-100	-100	-100	-100	-100	-100	-100	1	100	-100	-100	-100	1	-100	-100	-100	-100	-100
9	-100	1	-100	-100	-100	-100	-100	-100	1	-100	100	-100	-100	-100	-100	-100	-100	-100	-100	-100
10	-100	-100	-100	-100	-100	-100	-100	-100	1	1	100	-100	-100	-100	-100	-100	1	-100	-100	-100
11	-100	-100	-100	-100	1	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100
12	-100	-100	-100	1	-100	-100	-100	-100	-100	-100	-100	-100	-100	1	-100	-100	-100	-100	-100	-100
13	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	1	-100	-100	-100	-100	-100	-100	-100
14	-100	-100	-100	-100	-100	-100	-100	-100	1	-100	-100	-100	-100	-100	-100	1	-100	-100	-100	-100
15	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	1	-100	1	-100	-100	-100
16	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	100	-100	-100	-100	-100	1	-100	-100	-100	-100
17	-100	-100	-100	-100	-100	-100	-100	1	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	1	-100
18	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	1	-100	1
19	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	-100	1	-100

Figure 3: R-matrix as per definition in section 2.2

2.4 Choose a policy

2.4.1 Epsilon greedy policy

It is important to ensure that the agent explores the environment thoroughly before it starts exploiting the actions that will ensure the maximum returns in the long run. Through this policy the agent will select a random step to explore the environment with probability ε and a exploitation step that maximizes the Q value given the given current state, with probability 1-ε. ε value is initialized to 1. After each step by the agent, an epsilon decay factor is multiplied with the epsilon value so that the agent moves from exploration to exploitation as time progresses.

2.4.2 Random Policy

Here, Epsilon (ε) is set to 1 and an attempt is made to observe the behaviour of the agent when it chooses its actions randomly for each step in an episode.

2.5 Set the parameter values for Q-learning

2.5.1 Alpha – Learning Rate

The learning rate determines to what extent would newly acquired information by the agent be used to update the Q matrix and typically has a value between [0,1]. A factor of 0 makes the agent learn nothing (exclusively exploiting prior knowledge), while a factor of 1 makes the agent consider only the most recent information (ignoring prior knowledge to explore possibilities). We will run experiments with values 0.2,0.4,0.6,0.8,0.9.

2.5.2 Gamma – Discount Factor

The discount factor determines the influence of future rewards on Q value calculation and typically has a value between [0,1]. If the discount factor is near 0, then future rewards do not influence the Q value as much as the current reward. While if the discount factor is near 1, then future rewards will influence as much as immediate rewards. We will run experiments with values 0.2,0.4,0.6,0.8,0.9.

2.5.3 Epsilon

Epsilon determines the probability by which the agent performs exploration and exploitation. It typically has a value between [0,1]. A higher value ensures that the exploration will be longer. A value below 0.5 will make the agent exploit more. The epsilon decay factor plays a role in the decay of the epsilon value in an epsilon greedy policy. When epsilon >=0.5, decay rate is 0.99999 else it is 0.9999. This ensures that at the beginning, the agent explores more, while later as it learns, it starts to exploit.

2.6 Show how the Q-matrix is updated in a learning episode

We will demonstrate how Q matrix is updated for an episode in the start i.e. when Q matrix is all 0. Epsilon is taken as 0(in this example) so that we can demonstrate it deterministically. Since it is the beginning, the greedy policy will be equivalent to random, as all Q values are 0. The episode converges when it reaches the destination, which is reaching node 10. Assuming the following parameters: alpha=0.8,gamma=0.8. For **Experiment 1** -

i. Let us assume that start state is position 9. From position 9, ideally it should only go to 9->{1, 8, 10} as per the state transition definition. But here as the agent has only started Q matrix(all 0's)and the state transition is not enforced, but a result of the agent learning the optimal path based on the defined rewards. We can see that the agent randomly picks state 18 from the nodes ,as there are multiple nodes with same Q value as 0. Replacing eqn 1 values.

$$Q_{New}(9,18) = (1-0.8)*0 + 0.8 *(-100 + 0.8*0)$$

$$Q_{New}(9,18) = 0 + 0.8*(-100 + 0.8) = -80$$

ii. From state 18, the agent randomly picks state 10,

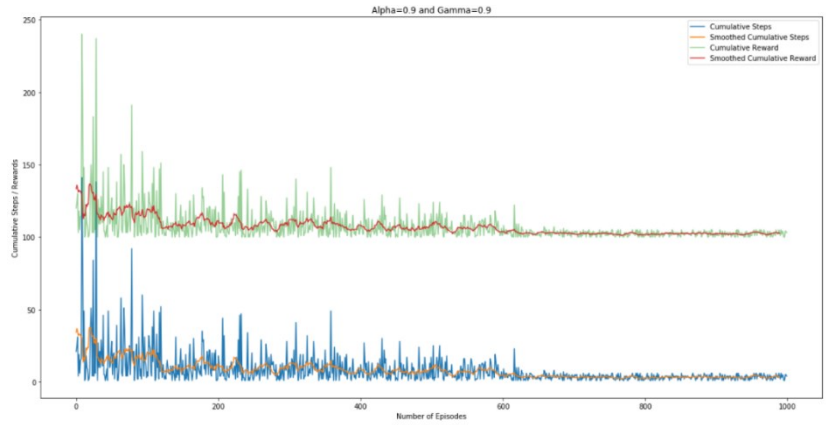
$$Q_{New}(18,10) = (1-0.8)*0 + 0.8 *(-100 + 0.8*0)$$
$$Q_{New}(18,10) = 0 + 0.8*(-100 + 0.8) = -80$$

iii. Now since the agent has reached the destination, it converges, and the episode ends.

For **Experiment 2** – agent randomly picks 10, so $Q_{New}(9,10) = (1-0.8)*0 + 0.8 *(100 + 0.8*0) = 80$.The Q matrix for the episodes is displayed in Jupyter notebook lasts section - 'Experiment to observe Q value change - Demo of delta learning and update rule'

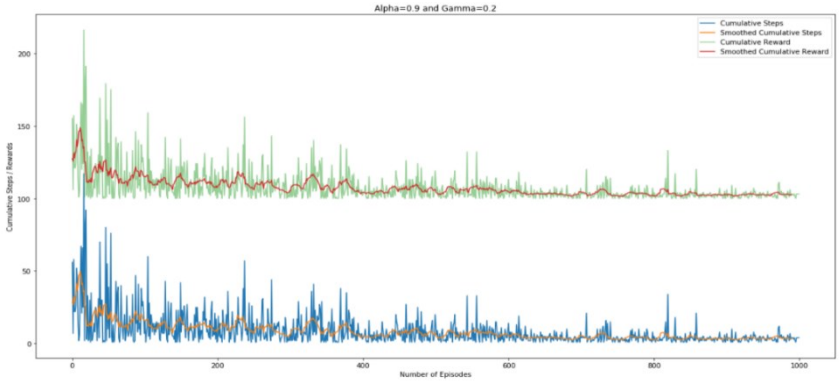
2.7 Represent performance vs. episodes & Repeat the experiment with different parameter values

Here we highlight the key experiments for different parameters and their impact on agent training. Only few are covered in the report due to space limitations. More examples are in the notebook.



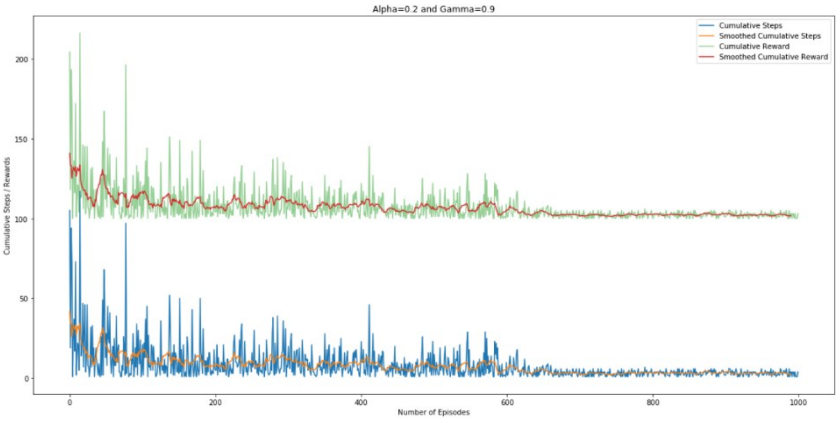
Alpha 0.9 and Gamma 0.9

The convergence is optimal taking around 630 episodes. High value of alpha and gamma ensures that the learning and discount of future rewards are factored in the agent's learning.



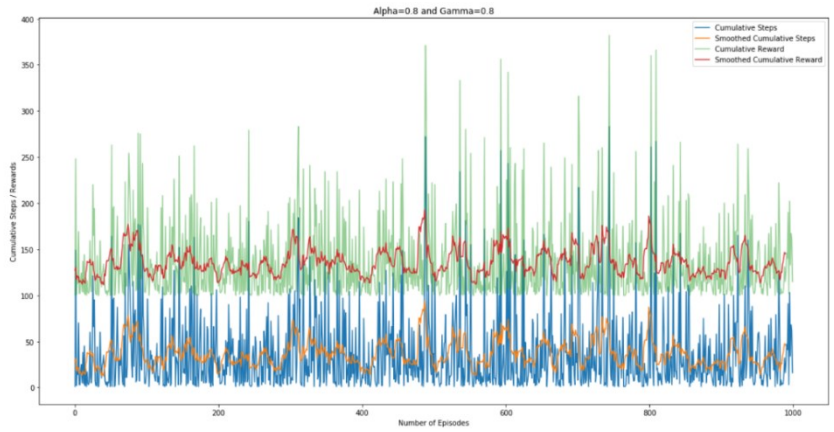
Alpha 0.9 and Gamma 0.2

The convergence appears to be slower when the gamma rate is low. Thus the memory of number of steps into the future that influences the learning is key for this problem.



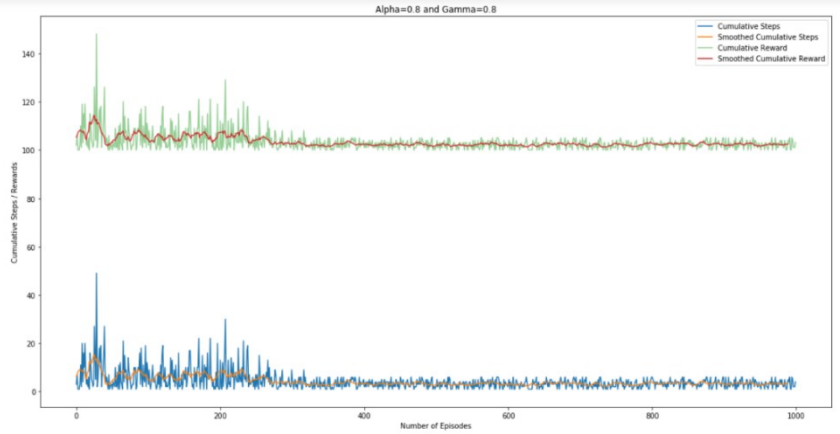
Alpha 0.2 and Gamma 0.9

The convergence appears to be like the first case with high values for both alpha and gamma. Thus based on the first 2 experiments we can reasonably conclude that gamma is more significant here in this environment.



Different policy - Random

There is clearly no convergence when attempting the random policy. Justifying the need for an epsilon greedy policy to solve this reinforcement problem.



Different Epsilon Start 0.6

We can see that there are early convergences on smaller epsilon value suggesting that the agent needs a shorter exploration phase for this environment because of its simplicity.

2.8 Analyse the results quantitatively and qualitatively

The quantitative experiment results for each parameter combination is available in the Q values sheet. The key results with min-max for metrics where the agent was able to solve the environment in the least mean steps per episode and average rewards per episode is highlighted. Key results below.

Alpha	Gamma	Epsilon start	Avg. Steps per episode	Avg. Rewards per episode
0.8	0.4	0.99	22.297	121.095
0.2	0.2	0.99	26.878	101.159
0.6	0.8	0.55	6.753	96.271
0.4	0.2	0.55	8.909	85.206

Based on the experiments we see that reduction of discount rate, makes the agent to take longer to converge and thus increases the number of episodes to learn the shortest path. The influence of gamma seems more than alpha for convergence. The best performance was for alpha 0.6, gamma 0.8 with average steps per episode of 6.75. As expected low values for alpha and gamma made the agent to take longer to converge. Changes to the epsilon value had the most drastic effect in terms of early convergence. For epsilon of 0.55 it took less than 300 episodes, which suggests that the agent need not have a long exploration phase and can quickly start exploiting. On attempting the random policy , as expected we observe that the agent cannot converge to the optimal shortest path.

We can observe that with enough training episodes for epsilon greedy policy, the agent always converges to optimality and learns the shortest path to the destination. Few samples:

shortest_path(19,10) -> [19, 18, 17, 7, 5, 1, 9, 10] & shortest_path(13,10)->[13, 12, 3, 2, 1, 9, 10]

3. Advanced – Deep Q Networks

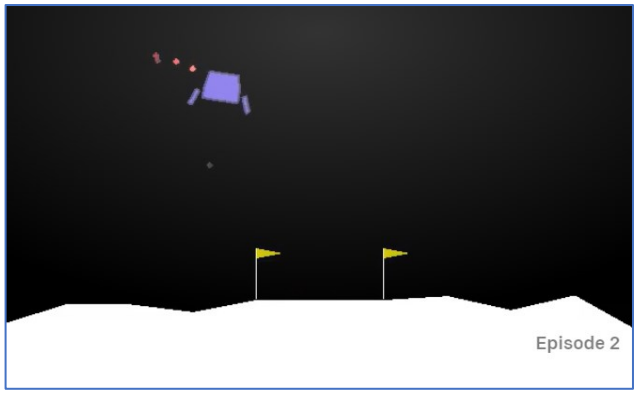


Figure 4 : Lunar Lander

3.1 Motivation and Expectation

Next we will attempt to solve the LunarLander-v2 environment from continuous control tasks in the Box2D simulator in Open AI Gym [9]. The goal is to teach the agent lunar lander to manoeuvre and successfully land on the moon in between the flags. Q learning does not work well within environments that have continuous states and action spaces. The Q-table would be theoretically infinite or very large even if discretized, and it would take a long time to converge to the true Q-values. Thus to solve the task of landing the lander in this environment, with continuous state space we will evaluate and compare two Deep Q networks (DQN) – DQN with fixed target, exploration and experience replay and Duelling DQN. DQN uses neural networks to approximate Q (s,a). We will first define the environment.

State space: The state space consists of eight-dimensions (6 continuous values + 2 discrete) with information on the lander's horizontal coordinate, vertical coordinate, horizontal speed, vertical speed, angle, angular speed and flags on whether the left and right legs touch the land.[9]

Action space: The action space has four actions available: do nothing, fire left engine, fire main engine, and fire right engine.[9]

Reward: Reward for moving from the top of the screen to landing pad and zero speed is about 100..140 points. If lander moves away from landing pad it loses reward back. Episode finishes if the lander crashes or comes to rest, receiving additional -100 or +100 points. Each leg ground contact is +10. Firing main engine is -0.3 points each frame. Solved is 200 points [9]

3.1.1 Deep Q Learning with Experienced Replay and Fixed Target Algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Figure 3: Algorithm [2]

Implemented Techniques

Experience Replay

If neural network training is done using each successive experience of the agent, due to high correlation between 2 consecutive states, training will be very instable. Experience replay is a technique used to add stability to training[2]. A fixed number of experiences (State, Action, Reward, Next State, Done) are stored in a replay buffer as the agent explores its environment. During the training process a random batch is sampled from the buffer. This helps reduce the correlations between the experiences in a training batch, which subsequently helps training .

Separate Target network

In addition to the network for generating current state Q value, a separate target network is used to generate the target Q value (temporal difference target) for the neural network training. The weights of the target network responsible for calculating the value of the state reached as a result of an action are frozen, and only periodically copied over from the online network. This adds more stability to training and quicker convergence[2]. Some variants do a soft update by retaining the existing weights by a factor tau(less than 1) and add the online network's weights multiplied by (1-tau).The max operator in DQN with fixed target uses the same values for action selection and evaluation. This makes it more likely to select overestimated values. Double DQN is an alternative.

3.1.2 Duelling Deep Q Network (DDQN)

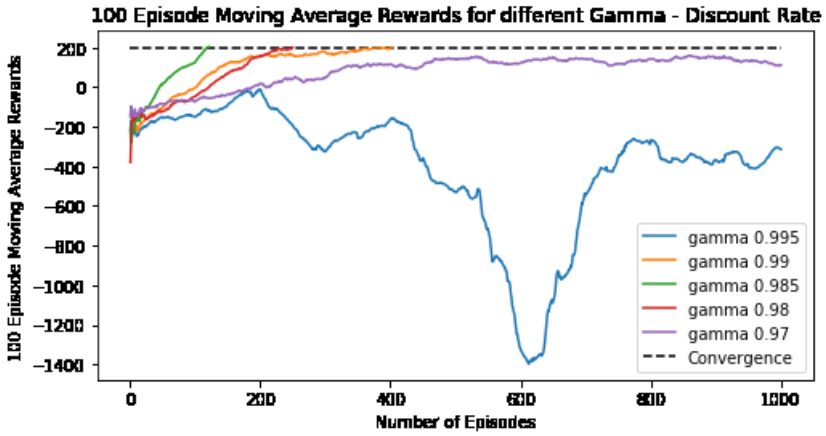
The duelling network has two separate estimators: one for the state value function and one for the state dependant action advantage function [4]. The duelling network learns which states are valuable, without explicitly learning the effect of every action for each state. Q(s,a) implicitly calculates the sum of: V(s): the value of being at a state and A(s,a): the advantage of taking a particular action at the state. It combines the results from the two estimators and estimates the Q value. DDQN thus learns value of a state for efficiently. This is particularly useful when the agent is in a state where its actions do not have any relevance to meeting the objective. It is redundant to calculate the value of each action in such a scenario. In the lunar lander environment for example, if the lander is on the left side of the sky, then the action to fire the left side engine is irrelevant. This architecture helps us accelerate the training as learning on the state's value will begin even if only one among the actions has been taken so far during training. We will do few experiments using duelling networks. It is expected that duelling network performs better than Deep Q network with fixed target, due to its faster training. The neural network is set up as a feed forward network with 2 hidden layers for each stream.

3.2 Quantitative Analysis varying parameters

We will target average of 200 points for 100 episodes as condition for convergence and stop further training on reaching it. If an agent does not converge it will be stopped at 1000 episodes. Identifying the right combination of hyperparameters so that the results would converge was very challenging. We had to run multiple experiments with combinations of **gamma, learning rate, epsilon decay, target model update frequency, batch size, replay memory buffer size, network hidden layer size** , **number of hidden layer neurons**. Throughout the experiments, we notice that for many combinations, the agent can cross 200 points often, but is not able to meet the convergence criteria of maintaining the average for 100 episodes continuously. Another observation is the dependence on the performance on initial weights for the network which results in different results each time during a run. First let us see the experiments for Deep Q network with fixed target (DQN- FT).

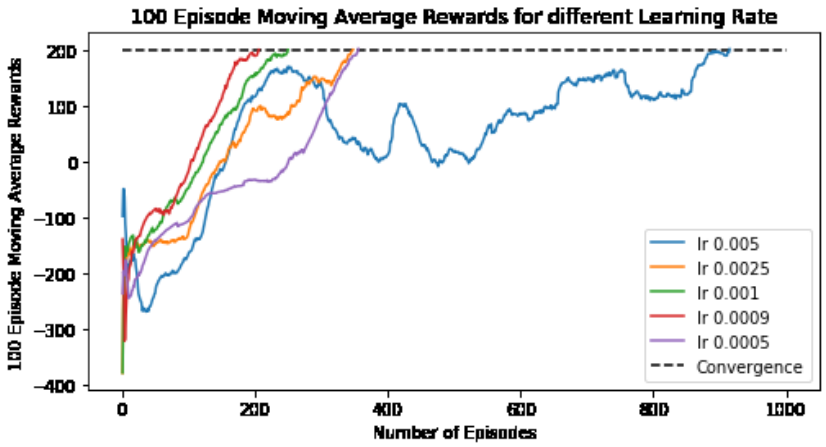
3.2.1 DQN- FT for Different gamma values

We ran the first experiment to observe the performance for different discount factor- gamma values = 0.995,0.99,0.98,.97. These values were arrived at by multiple trials observing the moving average pattern to determine the best possible values. The best values come within the top-5 among the published leader board results and is achieved in 120 episodes.



Gamma appears to be the most important parameter as once the best value for it was fixed, rest of the other attributes were easier to identify. Here we can observe that the agent converges the earliest when the gamma value is 0.985. For gamma value 0.98 and 0.99, the learning is smoother. For gamma 0.97 the agent almost converged but disappointedly could not. For each of the subsequent experiments a fixed value of gamma 0.98 was taken.

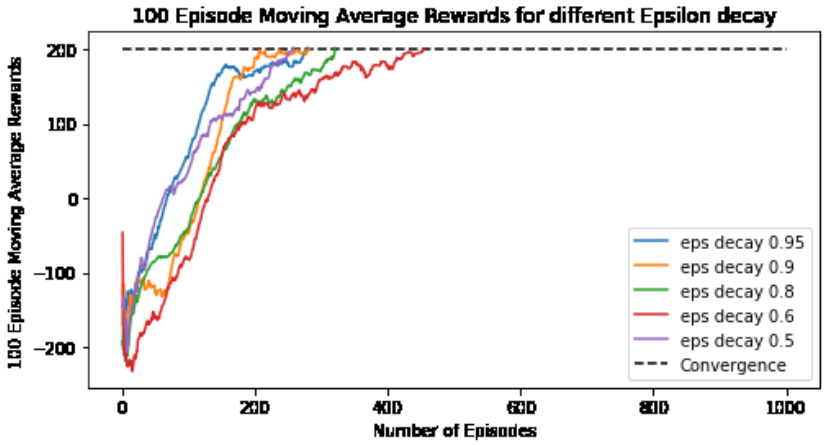
3.2.2 DQN-FT for Different learning rate values



The experiments for learning rate in the neural network show the best performance is for values close to 0.001.

3.2.3 DQN- FT for Different Epsilon Decay rate values

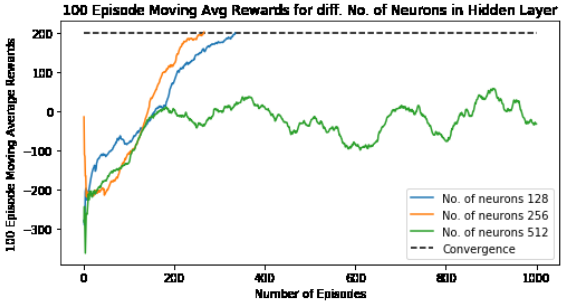
From the epsilon decay experiment we can see that even with very low exploration probability, the agent is able to sole the environment. This reinforces the importance of gamma – discount factor in the overall algorithm

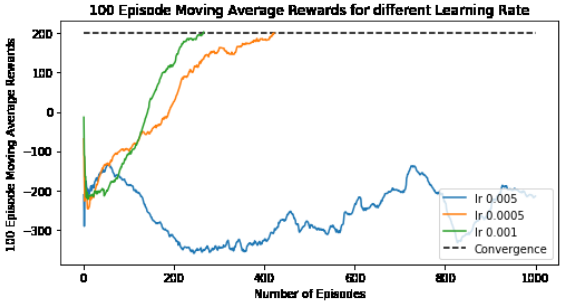


Other potential experiments could be network size, size of replay memory, update rate, etc. Due to time constraints we will not proceed with those.

3.2.3 Duelling DQN

Unexpectedly, the convergence for DDQN was only close and not improved compared to Deep Q Network with fixed targets. On average the best performing agents needed 200 + episodes to converge. Results from other experiments in the notebook.





3.3 Qualitative Analysis

We have successfully performed multiple experiments using Deep Q network with fixed target. Each of the parameters have a very short range of viable results which converge. To successful learn to land the lander, gamma must be kept high (>.97), so that the agent learns the reward in the future which are after many states. But a very high value ~1, adds too much noise to training which prevents convergence. Learning rate -the most important hyper-parameter to tune a neural network as it controls how much we are adjusting the weights of our network for each training epoch came close to 0.001 for the best performance. Large values cause instability in training while very low values prevent any learning and thus prevents the agent from converging. Increasing the learning rate also slows down training. Epsilon decay rate experiment hints that the agent can lean with a low exploration rate successfully. This alludes to moderate complexity and much lesser compared to a Atari environment. Since the other parameters -gamma and learning rate were already tuned for this experiment, we cannot conclude this with certainty. The update rate of the target network was kept as once every 50 steps. This contributed a lot to convergence and was a prerequisite before conducting the experiments. This reinforces the theory of introducing the fixed target and keeping the weights constant for few training epochs as it adds training stability and aids in convergence.

Duelling does not significantly improve the learning performance compared with Deep Q network with fixed targets . The results suggest that the efficient learning using the duel networks might not have a

real effect in improvement for this learning environment. A more complex environment may better show the significance of duelling.

4. Conclusion and Future Work

We have successfully represented determining the shortest path in a transport network as a reinforcement learning problem and solved it using Q learning algorithm. Analysing the performance over different parameter values helped to understand the influence of alpha, gamma , policy and epsilon on the agent's training in the environment. We have also implemented Deep Q learning with experience replay and fixed target and Duelling on the LunarLander environment from Open AI Gym. Future work would be to extend the quantitative analysis for advanced deep q learning techniques to size of replay memory, target update rate, batch size etc which are used for training are hyperparameters can be tuned for optimal performance.

For the advanced task, unexpectedly, the performance of duelling was not better than deep q network with experience replay and fixed target. This may be attributed to two reasons. The presence of an action - do nothing may have nullified the evaluation of an irrelevant state and that the duelling Q network algorithms is slightly more complicated due to the additional value stream also could have contributed. As such, based on Occam's razor, on a less complex environment like the LunarLander environment, the benefits of Duelling Q over Deep Q network with experience replay and fixed target may not be realized. However, in complex environments like Atari environments, it has been shown that duelling architecture has a benefit[4]. This could be another future work.

5. References

[1] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

[2] Mnih, Volodymyr, et al. "Playing Atari with Deep Reinforcement Learning." arXiv preprint arXiv:1312.5602 (2013).

[3] Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236

[4] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. 2016. Duelling network architectures for deep reinforcement learning. In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16). JMLR.org, 1995–2003.

[5] Tutorials and lectures from INM 707.

[6] https://theaisummer.com/Taking_Deep_Q_Networks_a_step_further/

[7] https://github.com/philtabor/Youtube-Code-Repository/tree/master/ReinforcementLearning/DeepQLearning

[8] https://deeplizard.com/learn/video/xVkPh9E9GfE

[9] http://gym.openai.com/envs/LunarLander-v2/

[10] https://www.python-course.eu/networkx.php

6. Appendix

All scripts used for this coursework are included as Jupyter Notebooks in the submission.