Implementing Deep Q-Networks on the Lunar Lander Environment

Sam Weinberg[1]

Abstract— **The Deep Q-Network (DQN) algorithm has risen to popularity in recent years by combining the initial Q-learning algorithm with a powerful neural network function approximator. Further contributions include Double Deep Q-Networks (DDQN) that addresses DQN's tendency to overestimate action values, and the dueling network architecture, which benefits learning in states with inconsequential actions. In this paper, we implement DQN and its variations within the LunarLander-v2 environment from OpenAI Gym. All algorithms were successfully able to learn the task of landing the shuttle smoothly in the landing zone. We report slight action value overestimates by the DQN, resulting in longer training time. The dueling network with 'maximum operator' method performed best, achieving the highest average reward among the algorithms through 50 tests. It also achieved this reward in the least amount of average steps per episode with the lowest standard deviation, indicating efficiency, consistency, and robustness to poor starting conditions. The benefits of each algorithm were reproduced in this environment, albeit to a lesser degree than in the Atari 2600 domain.**

I. INTRODUCTION

Reinforcement learning (RL) is a sub-component of machine learning which has demonstrated exciting potential to learn difficult tasks from scratch. The goal for an agent is to maximize the cumulative reward by learning a policy that optimizes its actions based on the surrounding environment [1]. Algorithms can be broadly divided into two classes: model-based and model-free. The model-based approach uses a learned or prior informed dynamics model to generate simulated experience with the environment. The simulated experience is used to improve the policy, but the results can suffer from model bias and render useless policies in practice. Model-free learning avoids this problem by obtaining experience from interacting directly with the real system. However, without a model to guide the policy search, model-free algorithms tend to be less sample-efficient.

One of the most prominent classical examples of a model-free algorithm is Q-learning, a foundational method that invokes ideas from dynamic programming and monte-carlo methods [2]. The algorithm was an RL breakthrough, but failed to solve more intricate problems with higher dimensionality due to poor scalability and the lack of computational power available. Over the past decade, incredible progress has been made in the area of neural networks, which have been applied to domains including pattern recognition (ImageNET, MNIST), natural language processing (IBM Watson, Google Cloud, etc.), and video games. DQN leverages these advances in neural networks to improve upon the tractability of implementing Q-learning. It uses a deep neural network as a nonlinear function approximator to estimate Q-values at different state-action pairs. Two key innovations were introduced in this algorithm to increase stability and data efficiency: the target network and experience replay. The success of this algorithm has been demonstrated in the Atari 2600 domain where it surpassed human-level performance using only frame pixels and game scores as inputs [3].

Further improvements have been made in recent years. It has been shown that Q-learning can overestimate Q-values because the maximum operator uses the same network to select and evaluate an action [4]. Double DQN (DDQN) introduces a second network, so that the action selection and evaluation can be performed on different networks. This mitigates the bias as demonstrated by the improved scores due to reduced Q-value estimates in several Atari 2600 games [5]. The dueling network architecture was introduced to address the idea that in several states the choice of action is inconsequential, and therefore, the Q-value does not need to be evaluated. This algorithm was able to further the state-of-the-art in the Atari 2600 domain and learn correct actions more quickly than its predecessors [6].

In this paper, we implement DQN, DDQN, and dueling DQN within the LunarLander-v2 discrete action space environment from OpenAI Gym [7]. The goal was to gain familiarity with the DQN algorithm, understand the benefits of its variations, compare the performance of its variations across several metrics, and qualitatively examine the different strategies produced by each algorithm.

II. BACKGROUND

For this problem, we consider a Markov Decision Process (MDP) wherein an *agent* interacts with an *environment*. The agent at a discrete timestamp t, observes a current state $s_t \in S$, and performs an action $a_t \in A$. The environment returns a reward r and a new state s_{t+1}.

The goal of the agent is to obtain the maximum discounted reward

$$R_t = \sum_{i=t}^{T} \gamma^{\,i-t} r_i \qquad (1)$$

where $\gamma\ \varepsilon\ [0,1]$ is the discount factor that determines the trade-off between current and future rewards.

A. Q-Learning

An agent's decision making process is defined by a *policy* $\pi : S \to P(A)$ which maps a state to a probability distribution over a set of possible actions. The state-action value function,

*This work was not supported by any organization

[1]Sam Weinberg is with University of Toronto Institute for Aerospace Studies, University of Toronto, Toronto, Canada sam.weinberg@mail.utoronto.ca

or Q function, describes the expected reward for a given state-action pair (s,a) and policy π.

$$Q^{\pi}(s,a) = \mathbb{E}\ [R_t \mid s_t = s,\ a_t = a,\ \pi] \tag{2}$$

Using dynamic programming, this equation can be solved recursively via the Bellman equation

$$Q^{\pi}(s,a) = \mathbb{E}_{s'}\left[r + \gamma\, \mathbb{E}_{a'}\ [Q^{\pi}(s',a')] \mid s,\ a,\ \pi\right] \tag{3}$$

The optimal Q-function is given by $Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a)$, which also satisfies the Bellman equation as follows

$$Q^{*}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'}\ [Q^{*}(s',a')] \mid s,\ a,\ \pi\right] \tag{4}$$

Q-learning is a model-free reinforcement learning algorithm, meaning it learns directly from experience interacting with the environment. The idea is to generate episodes using a policy allowing the agent to visit state-action pairs and build up a Q-table of values for each pair. At each iteration, the Q-values are adjusted based on the following update equation with learning rate α [2].

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\ [r_t + \gamma \max_{a'}\ Q(s_t',a_t') - Q(s_t,a_t)] \tag{5}$$

The optimal policy is simply the action that produces the highest Q-value in each state. Note that Q-learning is an off-policy algorithm, which learns about the optimal greedy policy without necessarily following it during the episode generation.

B. Deep Q-Learning

For many interesting problems, generating these Q-tables is intractable due to the high dimensionality of the state and action spaces. Neural networks are used as powerful nonlinear function approximators to estimate the optimal Q-function $Q(s,a|\theta_i)$ for a state-action pair. These Q-networks are parameterized by the weights θ_i at iteration i. The loss function used to train the Q-network is given by

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')}\left[(y_i^{DQN} - Q(s,a|\theta_i))^2\right] \tag{6}$$

where the targets are given by

$$y_i^{DQN} = r + \gamma \max_{a'} Q(s',a'|\theta_i^{-}) \tag{7}$$

DQN's success is derived from two key innovations that address existing problems regarding stability when using neural networks as a function approximator. First, the target network and online network are segregated, and the parameters from the online network θ_i are copied to the target network parameters θ_i^{-} every C steps. This introduces stability to the algorithm by fixing the targets for a set amount of time [3]. Alternatively, we can use a soft update that slightly but frequently updates the weights of the target network with $\tau << 1$

$$\theta^{-} \leftarrow \theta\tau + \theta^{-}(1-\tau) \tag{8}$$

Secondly, the algorithm stores the agent's experiences $e_t = (s_t,a_t,r_t,s_{t+1})$ at each timestep, and draws minibatches of experience uniformly from a distribution $D_t = \{e_1,...,e_t\}$ when training the Q-network. This technique inspired by biological learning is called experience replay and has the effect of reducing correlations between the samples used in training. The loss function is then adjusted as follows

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim D}\left[(y_i^{DQN} - Q(s,a|\theta_i))^2\right] \tag{9}$$

C. Double DQN

The double deep Q-network algorithm improves upon DQN's tendency to overestimate action values. The cause is attributed to the fact that the maximum operator in (7) uses the same Q-network to select the action and evaluate it [5]. This is explicitly represented as follows

$$y_i^{DQN} = r + \gamma Q\left(s', \arg\max_{a'} Q\left(s',a'|\theta_i^{-}\right)|\theta_i^{-}\right) \tag{10}$$

DDQN learns an additional function with weights θ', so that the action is selected and evaluated using different networks. This decouples action selection and evaluation, resulting in a fair evaluation of the policy which reduces overestimation. The DDQN targets are given by

$$y_i^{DDQN} = r + \gamma Q\left(s', \arg\max_{a'} Q\left(s',a'|\theta_i\right)|\theta_i^{-}\right) \tag{11}$$

D. Dueling Network Architecture

The dueling architecture was introduced to quickly evaluate policies in states with inconsequential actions. To give a brief outline of this network, we introduce the advantage function as the difference between the Q-function and value function

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{12}$$

The advantage function quantifies the relative benefit of each action. The dueling architecture proposes a duel-streamed network that estimates the value function $V(s|\theta,\beta_1)$ and advantage function $A(s,a|\theta,\beta_2)$ separately in the final hidden fully-connected layer using parameters β_1 and β_2 respectively. Figure 1 displays the initial single streamed DQN network compared to the architecture presented by

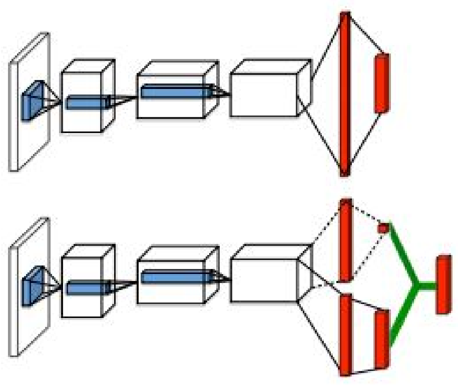
Fig. 1. Excerpt from Wang et al. comparing the single stream (upper) and dueling network architecture (lower). Two streams are used to separately estimate state-value and advantage function values. They are recombined using equation (13) or (14) to output Q-values [6].

Wang et al. where the algorithms were applied to the Atari 2600 domain [6]. They used convolutional neural networks (CNNs) for the hidden layers, and fully connected layers to compute the advantage and value functions. Note that in the LunarLander-v2 environment, we do not have image inputs and will only use fully connected layers.

The two streams can be combined together in two manners: 'maximum' or 'average'. To ensure that the state value function and advantage function are distinguishable upon aggregation, the advantage function stream is forced to zero at the chosen action. For an optimal action a^*, we then obtain $Q^\pi(s,a^*|\theta,\beta_1,\beta_2) = V^\pi(s|\theta,\beta_1)$.

$$Q^\pi(s,a|\theta,\beta_1,\beta_2) = V^\pi(s|\theta,\beta_1) + \Big(A^\pi(s,a|\theta,\beta_2) - \max_{a'} A^\pi(s,a'|\theta_i,\beta_2)\Big) \tag{13}$$

To increase stability during training, the maximum operator can be replaced by an averaging over all possible actions. This relinquishes the ability to uniquely identify the value and advantage function, but provides stability in that the average advantage value may change less quickly than the maximum.

$$Q^\pi(s,a|\theta,\beta_1,\beta_2) = V^\pi(s|\theta,\beta_1) + \Big(A^\pi(s,a|\theta,\beta_2) - \frac{1}{|A|}\sum_{a'} A^\pi(s,a'|\theta_i,\beta_2)\Big) \tag{14}$$

The dueling network architecture algorithm remains the exact same as DQN except for the duel-streamed vs. single streamed calculation of the Q-value.

III. EXPERIMENTAL SETUP

The LunarLander-v2 environment available from OpenAI Gym was selected to implement the DQN algorithm and its variations. The environment consists of a rough moon surface with a desired landing position located in the middle. The agent corresponds to a space shuttle with multiple thrusters used to control its motion. Figure 2 displays the LunarLander-v2 environment. In this figure, we denote 'up' as using the main thruster.

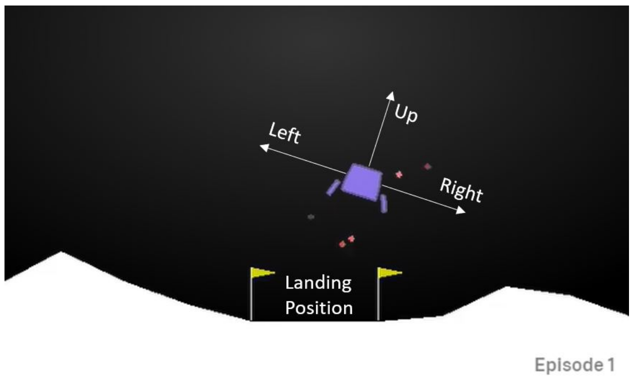

Fig. 2. LunarLander-v2 environment from OpenAI Gym. Agent has four discrete action choices (right engine, main engine, left engine, do nothing). The goal is to land the shuttle in the landing position in a smooth manner while using the main engine as little as possible.

The action space consists of four discrete options: Do nothing (0), left thruster (1), main thruster (2), or right thruster (3). The state vector is 8-dimensional, consisting of: position (x,y,ϕ), velocity $(\dot{x},\dot{y},\dot{\phi})$, and each leg contact with the ground as a boolean (r_{1-0},l_{1-0}).

The Q-network is designed to match the input and output specifications of the environment. The input layer consists of 8 units (9 with bias), and the output consists of 4 units representing a Q-value for each action. The architecture of the hidden layers was tuned to obtain the best results, and adjusted slightly for each variation of DQN. Using 128 neurons in each hidden layer was found to give reasonable results. Rectified linear unit (ReLU) activation functions were used in the neurons of all the hidden layers. Figure 3 displays the general architecture for the DQN algorithm implementation in this environment.

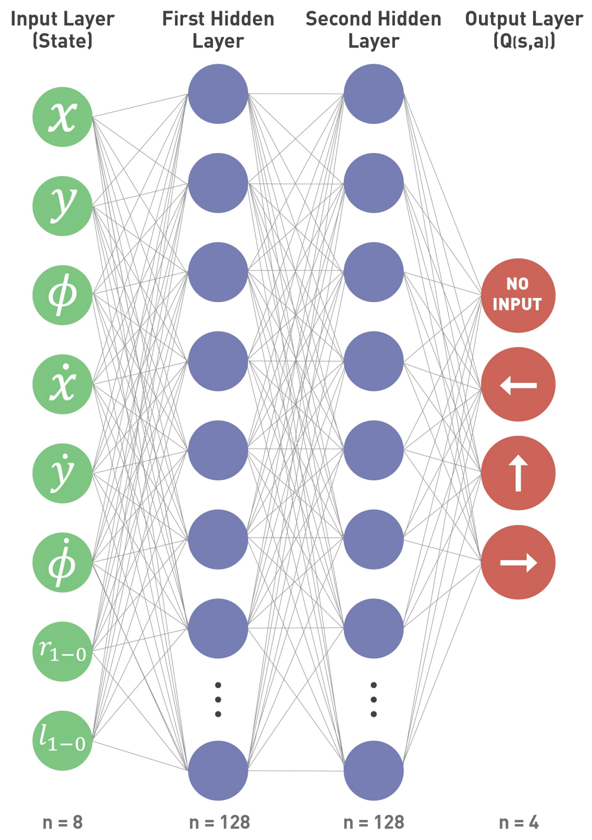

Fig. 3. Diagram displaying general neural network architecture used in the DQN algorithm. The input layer contains the 8D state, the two hidden layers each have 128 neurons, and the output layer contains the Q-value for each action.

The following excerpt from OpenAI Gym details the rewards and penalizations returned by the environment [7]:

- Landing pad is always at coordinates (0,0). Coordinates are the first two numbers in state vector. 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

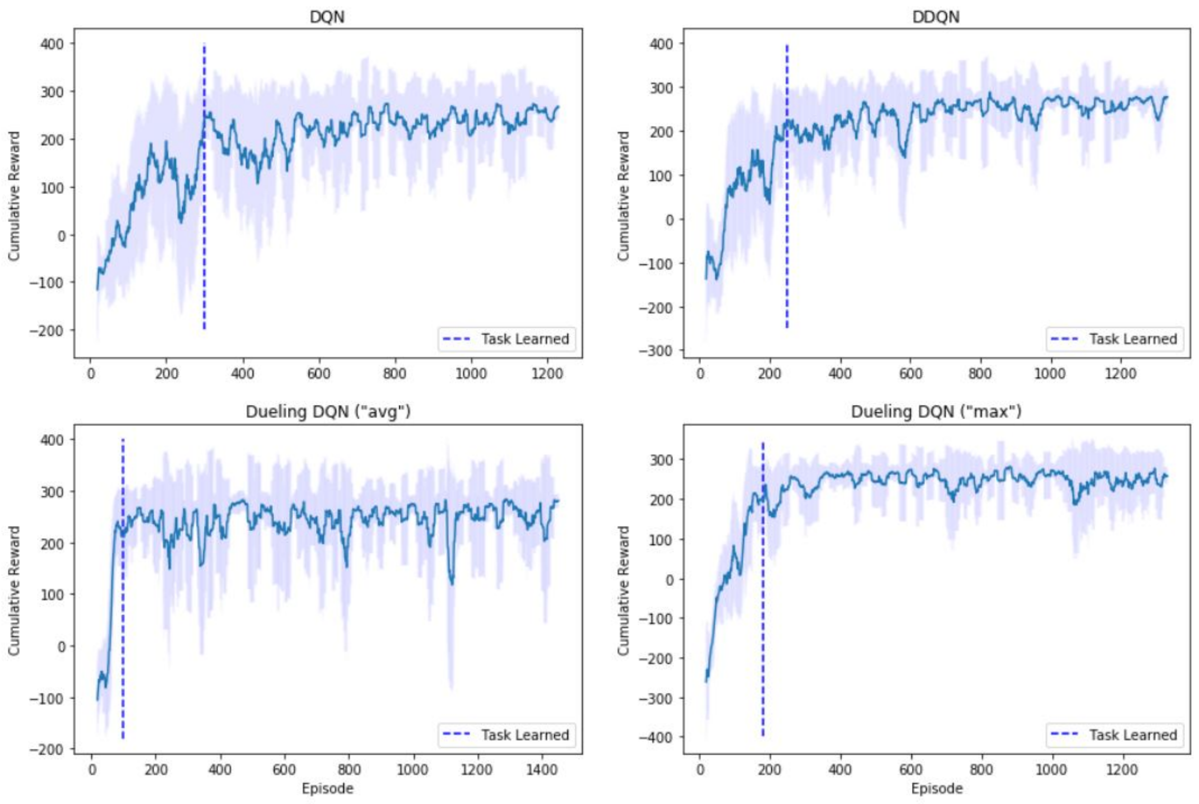

Fig. 4. Cumulative reward plots during training for each of the four algorithms. The dark blue line represents the rolling mean cumulative reward with a window of 20 episodes and the shaded area is one standard deviation. The dashed blue line represents the approximate number of episodes required to learn the task as defined by OpenAI Gym.

is +10. Firing main engine is -0.3 points each frame. Solved is 200 points. Landing outside landing pad is possible. Fuel is infinite, so an agent can learn to fly and then land on its first attempt.

The environment incorporates randomness so that the learned task is non-trivial. It uses random 'winds' as perturbations to alter the subsequent state, making the problem stochastic in nature. Additionally, it randomizes the initial conditions so that the lander must learn to gain control from slightly different positions. Without this, the algorithm would learn to repeat the trivial task of slowly landing from the same position.

IV. RESULTS

This section displays the results of the experiments. All algorithms were able to learn a policy that successfully solved the environment on a regular basis. The hyperparameters used in the experiments are displayed below in Table I.

TABLE I

HYPERPARAMETERS USED IN ALL OF THE EXPERIMENTS.

Hyperparameter	**Value**
1st Hidden Layer	128 Units
2nd Hidden Layer	128 Units
Activation Function	ReLU
Mini-batch Size	32
Replay Memory Size	1000000
Discount Factor γ	0.99
Target Update Rate τ	0.01
Learning Rate α	0.001
Exploration ε	0.1

The parameters were tuned for the original DQN algorithm, and the values were reused for its variations to obtain a fair comparison and demonstrate the best case scenario for DQN. Parameters were tuned qualitatively, as a formal grid search was found to be computationally infeasible given the hardware. An ε-greedy policy was used to allow for exploration during training.

Figure 4 displays the cumulative reward for each algorithm as a function of episode. A rolling average with a window of 20 episodes was used to smooth the data, and the shaded area is the standard deviation of the rolling average. As outlined by OpenGym AI, the environment is considered solved with a score of 200. The dotted blue line approximately represents the number of episodes required for each algorithm to learn the task.

In terms of stability, training speed, and ability to obtain an optimal policy, the plot reinforces the advantages of the improved DQN algorithms. The DQN algorithm performed the worst, learning the task in approximately 300 episodes but reaching a steady policy in 500 episodes. The DDQN improved upon the DQN results, potentially due to less optimistic action values which are discussed below. The results of overestimated Q-values are not nearly as detrimental to the DQN's performance as they were found to be in some Atari 2600 games [5].

The dueling networks outperformed the DQN and DDQN algorithms, learning the task in under 200 episodes. The ”averaging” method of combining the value and advantage function learned the task extremely quickly in approximately 100 episodes, but was prone to instability throughout the learning process. Additionally, it may have randomly explored high reward state-action combinations in the initial episodes as inferred from its relatively high starting point for the cumulative reward. Conversely, the 'maximum' method also learned extremely quickly and the policy remained stable throughout the training. This result is somewhat contradictory to the predictions of the dueling network, which postulate that the 'averaging' method should be utilised for its increased stability [6]. We also note that in an environment with a higher dimensional action space, we would expect the discrepancy between the DQN, DDQN, and dueling networks to increase as the dueling networks share state values across similar actions. In this environment, the action taken in most states is likely to be highly consequential.

The DQN and DDQN are directly compared to observe the impact of overoptimistic Q-value estimates. Figure 5 demonstrates DQN's tendency to overestimate, with the mean Q-value slightly higher than DDQN's throughout the majority of training. This coincides with lower cumulative rewards and less stable training as seen in the lower plot. The difference is much less significant than in the Atari 2600 domain [5], but the impact is noticeable when comparing the episodes required to learn the task.

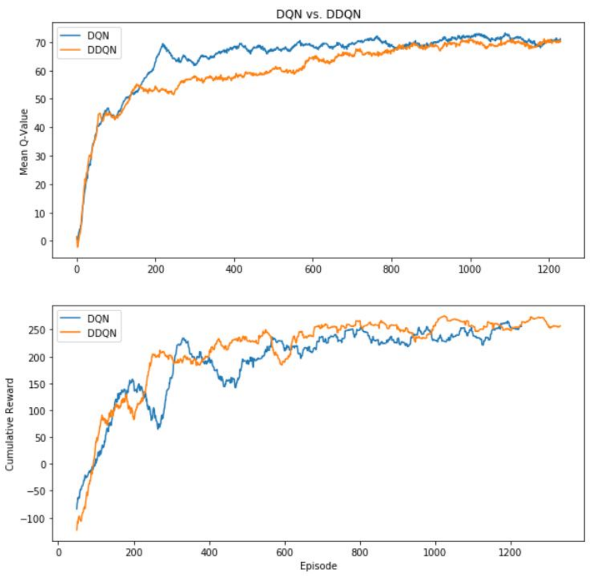

Fig. 5. A comparison of DQN and DDQN demonstrating the slight overoptimism of DQN and its impact on the training speed. The upper plot shows the mean Q-value from each episode. In DQN, the same network is used to select and evaluate the action, resulting in a positive bias. The lower plot shows how this effect coincides with lower episodic cumulative reward.

From a qualitative perspective, the algorithms learned different methods to achieve the maximum cumulative reward, some of which were more optimal than others. The DQN method was the least impressive as reflected in the test results in Table II. The algorithm was less flexible in its ability to overcome poor starting conditions, and it rarely landed in the center of the landing pad. The DDQN had a better response to poor starts, however it often landed with significant velocity resulting in lower reward. The dueling network with 'averaging' method had very similar behaviour to the DDQN. In each of these algorithms, the agent would often take a sub-optimal action (left or right engine) after landing, resulting in a maximum step (1000) episode. This increased the average amount of steps per episode and can also decrease the reward if the agent moved away from the target.

The algorithm that achieved the most stable and consistent optimal policy was the dueling DQN with 'maximum' method. The agent consistently landed smoothly in the center of the landing zone. It always selected the 'do nothing' action upon landing, resulting in low average episode steps and standard deviation. It used the main engine as little as possible at the beginning of the episode to reduce penalization, and then it recovered control to land with near-zero velocity. DDQN and the 'averaging' duel network also conserved main engine usage, but were unable to recover as smoothly to achieve maximal reward. We can postulate that the maximum advantage function over the possible actions was significantly higher than the average. In this environment, the majority of the states have highly meaningful actions that directly determine whether the agent lands smoothly or crashes. As such, the advantage stream's impact in consequential situations may have a greater impact on the aggregated Q-function when using the 'maximum' method. Upon landing, several actions (left, right, do nothing) have a similar impact due to the friction of the legs with the ground. As reported in Wang et al., the dueling network is more able to identify the correct action as redundant actions are added [6].

V. CONCLUSIONS

The DQN, DDQN and dueling DQN algorithms were applied to the LunarLander-v2 environment to understand their advantages and impact on the agent's behaviour. The results essentially reproduced the findings from the results in the Atari 2600 domain, with each algorithm performing better than the preceding. The magnitude of overoptimistic Q-values were minimal, yet their impact on the overall cumulative reward was considerable. The dueling architecture was shown to be beneficial for this environment, but its true potential would be better demonstrated in a high dimensional action space [6]. Ultimately, the problem was shown to be solvable by all algortihms.

To compare the computational efficiency of the algorithms would be somewhat meaningless given the conditions under which the training was performed. A CPU with i7 2.20GHz 6-core processor was used while running other applications and under different battery conditions. In a future work, we

TABLE II

TEST RESULTS FOR EACH OF THE DQN ALGORITHMS. 50 TESTS WERE RUN USING THE FULLY TRAINED ALGORITHMS. RESULTS SHOW THE AVERAGE CUMULATIVE REWARD AND NUMBER OF STEPS REQUIRED TO COMPLETE EACH EPISODE.

	Average Cumulative Reward	**Average Steps**
DQN	225.46 +/- (56.51)	406.68 +/- (291.35)
DDQN	256.90 +/- (39.93)	329.40 +/- (264.98)
Dueling DQN ('Avg')	262.42 +/- (36.11)	317.00 +/- (260.77)
Dueling DQN ('Max')	265.71 +/- (21.62)	243.80 +/- (27.63)

would obtain upgraded hardware (GPUs) to run the training under fixed conditions, and perform a formal evaluation of the computational efficiency of each algorithm.

Another innovation that has contributed to improved DQN results was *prioritized replay* as demonstrated by Schaul et al [8]. The idea was to use the temporal difference (TD) error as a proxy for the unexpectedness of a transition, and use it to determine the frequency with which the experience is replayed. This method of prioritizing experiences related to high learning progress has been shown to increase learning efficiency. Future work in the LunarLander-v2 environment would include implementing this technique.

OpenGym AI provides a LunarLanderContinuous-v2 environment which is identical to the one we used, but contains a continuous action space. This allows the agent to control the percentage of power used by the main thruster. Future work of this project would include extending our analysis and comparison into the continuous action domain. DQN cannot be applied directly to continuous action spaces due to an iterative optimization over discrete actions at each step. Lillicrap et al. leveraged ideas from DQN and deterministic policy gradient (DPG) [9] to create an algorithm termed deep deterministic policy gradient (DDPG) [10]. This algorithm utilises an *actor-critic* approach, where the actor-network $\mu(s,a|\theta^{\mu})$ maps a state to an action, and the critic-network $Q(s,a|\theta^{Q})$ is similar to that of the DQN. Target networks are used for both the critic and actor networks. Since the rewards/penalizations remain the same in both environments, it would be interesting to compare the results of the DDPG on the continuous environment to our discrete action space results.

In conclusion, this project has provided an excellent opportunity to learn about the intricacies of the DQN, DDQN, and dueling DQN algorithms, and implement them within a challenging environment. A video of the training results can be found at this link: `https://drive.google.com/drive/folders/1cYSnCvRwWMDPQcvq3KTMuAD28WmdwCsl`

REFERENCES

[1] R. Sutton and A. Barto, Introduction to Reinforcement Learning, MIT Press, 1998

[2] C.J. Watkins and P. Dayan, Q-Learning, Machine Learning, 8: 279–292, 1992

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg D. Hassabis, Human-level control through deep reinforcement learning, Nature, 518 (7540):529–533, 2015.

[4] H.van Hasselt, Double Q-learning. NIPS, 23:2613–2621, 2010

[5] H. van Hasselt, A. Guez, and D. Silver, Deep Reinforcement Learning with Double Q-learning, arXiv preprint arXiv:1509.06461, 2015

[6] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot N. de Freitas, Dueling Network Architectures for Deep Reinforcement Learning, arXiv preprint arXiv:1511.065812016, 2015

[7] Gym.openai.com. 2020. Gym: A Toolkit For Developing And Comparing Reinforcement Learning Algorithms. [online] Available at: https://gym.openai.com/envs/LunarLander-v2/ [Accessed 25 April 2020].

[8] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized Experience Replay, arXiv preprint arXiv:1511.05952 2016

[9] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, Deterministic policy gradient algorithms. In ICML, 2014.

[10] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971

[11] J. Buchli, F. Farshidian, A. Winkler, T. Sandy, M. Giftthaler, Optimal and Learning Control for Autonomous Robots, CoRR, arXiv preprint aXiv:1708.09342