OpenAI Lunar Lander Learning - Final Project

Selmo Almeida, Jason Peloquin, Jacob Reed
University of Colorado Boulder, Boulder, Colorado
Ann and H.J. Smead Aerospace Engineering Sciences
May 7, 2023

I. Introduction

The task of placing a lunar lander on the Moon has always been a challenging problem in the field of robotics and artificial intelligence. This is due to several factors including but not limited to low gravitational force, the lack of atmosphere, and the Moon having very uneven terrain. With recent advancements in reinforcement learning, there has been a growing interest in using these techniques to solve complex control problems. This paper explores the use of the Discrete Q-learning and Deep Q-Network (DQN) reinforcement learning algorithms to land a spacecraft within the OpenAI gym environment. The approaches involve training the agent to learn through trial and error, which allow the agent to learn from it's own experience and gradually improve its performance. To evaluate both techniques, learning curves and reward tracking is provided to compare each methods effectiveness. Thus, this paper shows the feasibility of using reinforcement learning to solve complex control problems and provide knowledge as well as limitations to each of these reinforcement learning algorithms for placing a lunar lander on the surface of the Moon in a simplified environment.

II. Background and Related Work

Lunar landings have been studied extensively in robotics and artificial intelligence communities [2] [3]. One of the biggest challenges is the high-dimensional state and action spaces, requiring sophisticated control strategies to successfully (and safely) navigate and land. Traditional control methods, such as model-based control, demands precise mathematical modeling of the system, which is often very complex and difficult to obtain. This can be computationally expensive and opens the door for Discrete Q-Learning, DQN, and other reinforcement learning algorithms to learn optimal policies directly from data without requiring a model of the system. The abilities of this category of algorithms has the potential to reduce the computational time required to learn complex environment layouts and make decisions based on high-fidelity, modeled reward structures. These learning strategies can be found across various applications and have been shown to be highly successful. From playing Atari games to controlling robotic arms, this field of study is showing how reinforcement learning algorithms can revolutionize how we approach dynamical systems across an extensive range of applications (playing games, controls, exploration, etc).

III. Problem Formulation

The problem to be addressed answers the question of *how can we safely touch down a lunar lander on the Moon's surface?* Due to the ever changing lunar landscape and unknown distribution of rewards, a heuristic policy was determined to not be a practical method for safely landing. Though, in contrast, it was found the designated landing location remained constant within the environment and was always set to the center of the generated lunar surface. Thus, for this project the implemented learning strategies would be required to seek out paths based on unknown rewards, with the highest reward designations based on quick and/or gentle touchdowns. Of course, there is additional complexity within the problem, which is how the lunar lander was inserted into the environment. This is seen as a random variable. Thus, solving this problem was broken into 3 levels of success to measure learning algorithm effectiveness.

1) **(Minimum Working Example)** Implement an algorithm such as Q-learning, double Q-learning, SARSA, or others from scratch and successfully train the lander to safely touch down on the Moon in a discrete environment. If this is seen as too simple, randomness will be injected into the environment in the form of turbulence or wind which would simulate imperfect burns and transition uncertainty.
2) **(Primary Goal)** Use deep reinforcement learning/neural networks to successfully place the lunar lander on the Moon's surface while tuning the neural network with different hyperparameters. These include but are not limited to optimizers, loss functions or convolutional layer complexity to contrast each network's performance.
3) **(Stretch Goal)** Utilize liquid neural networks to have the neural network composition change throughout the simulation. The dynamic morphing of neuron structure may improve the lander's reinforcement learning results. This approach may be successful due to the lunar lander having multiple flight regimes based on altitude. Furthermore, a static neural network may not be sufficient to optimize its landing abilities in each zone. This type of neural network was developed in recent years to address autonomous system and computer vision problems.

In addition to the levels of success above, some understanding of the State, Action and Reward structures was also required. For this particular problem, the OpenAI Gym Lunar Lander environment has the following characteristics [7]:

1) **Action Space**
   - 0: do nothing
   - 1: fire left orientation engine
   - 2: fire main engine
   - 3: fire right orientation engine
2) **Observation Space**
   - X position: [-90, 90]

- Y position: [-90, 90]
- X velocity: [-5, 5]
- Y velocity: [-5, 5]
- Angle from horizontal: [$-\pi$, π]
- Angular velocity: [-5, 5]
- Left leg in contact with ground (boolean): [0, 1]
- Right leg in contact with ground (boolean): [0, 1]

3) **Rewards**
   - increased/decreased the closer/further the lander is to the landing pad
   - increased/decreased the slower/faster the lander is moving
   - decreased the more the lander is tilted (angle not horizontal)
   - increased by 10 points for each leg that is in contact with the ground
   - decreased by 0.03 points each frame a side engine is firing
   - decreased by 0.3 points each frame the main engine is firing
   - -100 points for crashing, +100 points for landing safely

Based on the structure above, an episode was considered to be a solution if it scored at least 200 points.

IV. Solution Approach

The two selected learning strategies for this project are Discrete Q-learning as well as the use of a Deep Q-Network (DQN). The primary reasons for looking more deeply into these algorithms was based on flexibility, learning efficiency, generalization, as well as performance. In terms of flexibility, Discrete Q-learning and DQN can handle different types of state-action spaces, making them ideal for solving a wider range of problems. Additionally, Discrete Q-learning is preferred for a small and discrete state-action space, while DQN can handle continuous state spaces and discrete action spaces. In regards to learning efficiency, these two reinforcement learning algorithms are able to learn optimal policies through trial and error, which allows each of these algorithms to learn from it's own experiences and improve it's performance, which is ideal when solving complex problems such as lunar landings. Being that lunar landings may likely occur in new and unseen environments, these Discrete Q-learning and DQN algorithms are great when learning generalized policies that can be applied to never-before-seen environments, which is ideal in real-world scenarios.

In conjunction to the algorithms mentioned above, an ϵ-Greedy policy was applied to each due to it being a common exploration strategy. The benefit of using this policy within the Discrete Q-learning and DQN algorithms is that it balances exploration and exploitation. This policy is flexible and can be tuned to either be set to explore for n-steps and commit from then on, or be set to allow the agent to randomly explore the environment over the course of its training. In either case, this allows the agent to discover new actions and learn optimal policies.

V. Discrete Q-Learning

This reinforcement learning strategy was implemented from scratch written in Python and aims to learn Q values via an incremental estimation update, as seen in Equation 1. The estimate of Q in Equation 2 is based on finding what the maximum Q would be in our next state. The exploration policy chosen was the ϵ-Greedy policy seen in Equation 5. Given a state, the ϵ-Greedy policy chooses the action that gives the largest Q value state-action pair with probability $1 - \epsilon$, otherwise, it chooses a random action with probability ϵ [1].

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \quad (1)$$

$$\hat{Q}(s') = r + \gamma \max_{a'} Q(s',a') \quad (2)$$

$$\text{Temporal Difference (TD)} = \hat{Q}(s') - Q(s,a) \quad (3)$$

$$\rho = \frac{\text{number of wins}}{\text{number of tries}} \quad (4)$$

$$\text{Greedy Strategy: Choose} \begin{cases} \text{random} & \text{w.p. } \epsilon \\ \text{argmax}_a \, \rho_a & \text{w.p. } 1-\epsilon \end{cases} \quad (5)$$

In order to use this algorithm, the continuous observation space of the OpenAI Gym Lunar Lander Environment needs to be discretized. This discretization will undoubtedly lead to lower performance due to skewing the information about the lunar lander's true state. Each state was discretized into 8 values linearly spaced apart between their minimum and maximum values; except for the boolean observations, where each has 2 discrete values. The discrete Q-learning Python class has a method called `discretize_state` that takes a continuous state and finds the nearest discrete state.

For the algorithm implementation itself, an episode begins by resetting the environment, making an observation and discretizing it, selecting an action based on the ϵ-Greedy policy, taking a step in the environment using this action, and then making another observation and discretizing it. Then Equation 1-3 are used. The algorithm loops like this until the environment terminates (lander crashes), truncates (lander drifts outside of box), or the maximum amount of steps is reached and the environment is reset. All the episodes are run with the Q-table updating for each episode, then the Q-table is evaluated with $\gamma = 1.0$. This produces a score that shows the learning curve for the agent.

The following are the parameters used:

- $\gamma = 0.99$
- $\alpha = 0.01$
- $\epsilon = 0.1$
- $n\_episodes = 10,000$

VI. Discrete Q-Learning Results

Our group approached the use of this algorithm with the expectation it would perform poorly. We noticed that after a number of episodes, the lander would mostly hover near the top of the view box and eventually drift either to the left or right, outside of the viewable space. Figure 1 shows the performance of the Q-learning algorithm after 10,000 episodes. It learns fairly quickly, but then plateaus at a score of about -50. It is reasonable to believe the loss in information from the discretization of the observations along with the maximization bias present in the Q-learning algorithm negatively impacts lander results. If we were able to discretize the observation to more points or possibly implement double Q-learning to reduce maximization bias, then we may have been able get better results from a tabular method. Figure 2 shows the results of the best scoring episode from the Q-learning algorithm. The agent appears to passably learn how and where to land but it fails to land soft enough to prevent a crash.

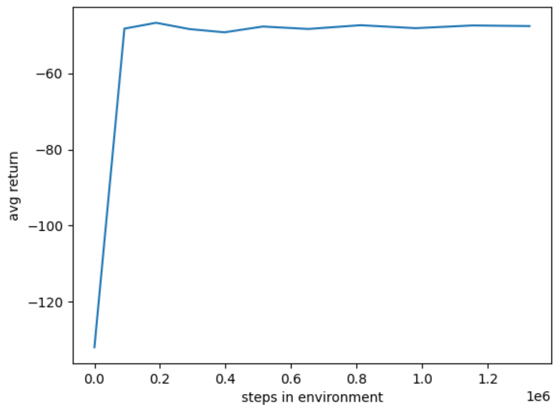


Fig. 1: Q-Learning Curve - 10k episodes

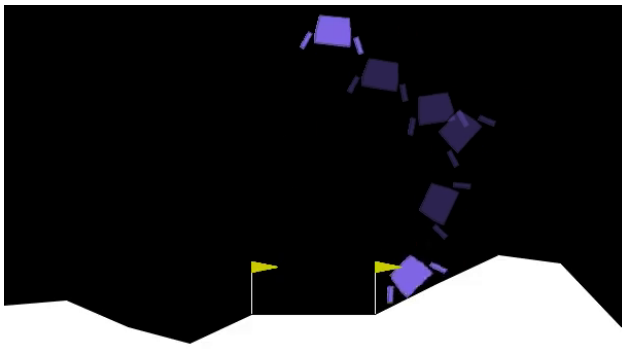


Fig. 2: Best Score Q-Learning Episode

VII. Deep Q-Network (DQN)

To overcome the inability of tabular Q-learning to successfully place the lunar lander on the Moon's surface in a discrete form, a deep Q network (DQN) approach was implemented to accommodate the default continuous state space environment. Equation 1 through Equation 5 are still utilized in the DQN approach but their propagation is facilitated through a neural network and tensor representation versus a tabular form.

*1) **DQN Structure**:* A baseline neural network was developed to promote deep Q-learning and hyperparameter tuning. The structure of the neural network to be trained is comprised of an input layer relying on the lunar lander's current state and 3 fully connected, linear, hidden cortex layers comprised of 200 neurons each. The outputs of the neural network are the 4 thruster actions available to the lunar lander. Located below are the original neural network parameters used as initial starting points for further optimization.

- 3 Fully connected hidden layers
- 200 Neurons per layer
- ReLU activation function
- 400 Episodes per epoch
- 1E-4 Learning Rate
- 1E6 Memory Buffer Size
- 1E3 Replay Buffer
- 500 Max Steps per trial run
- 1E-4 Epsilon Decay rate

*2) **Hyperparameter Analysis**:* Various hyperparameter combinations were explored to optimize the lunar lander problem, however, for brevity only 3 will be discussed in this paper due to their profound effect on performance. The hyperparameters in focus with this specific DQN structure include: the activation function, loss function, and replay buffer size. Each of these values have an impact on improving the performance of a DQN model with respect to successfully placing the lunar lander on the surface, agent learning rate, computational speed, and model stability. Note that finding the optimal hyperparameter values can be highly demanding, requiring careful experimentation and analysis as all of the neural network hyperparameters work in concert together. Adjusting one hyperparameter reverberates through all the other ”tuning knobs” of the model and can lead to unexpected changes in performance.

*3) **Activation Functions**:* For this approach, the ReLU and Tanh activation functions were selected. ReLU is currently the industry standard due to it addressing the issue of vanishing gradients which surface during back propagation. Additionally, in most cases the implementation of ReLU leads to faster convergence as it features a fixed linear slope in it's calculation and does not rely on exponentiation as Tanh does. Tanh, although outdated, is still an appropriate choice for hidden layers of a neural network. Tanh offers the advantages of ranging from -1 to 1 in it's output. This allows highly negative activation values that are produced in the neural network to persist versus ReLU, which features a lower bound of zero [5]. This wider range of values may lead the neural network to articulate neuron activation more delicately. Depictions of both activation functions are in shown in Figure 3, below.

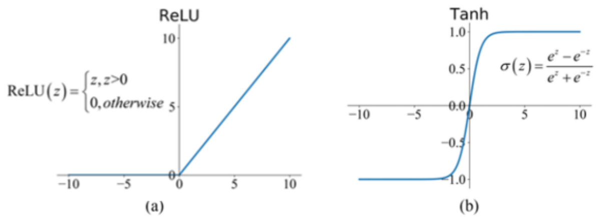
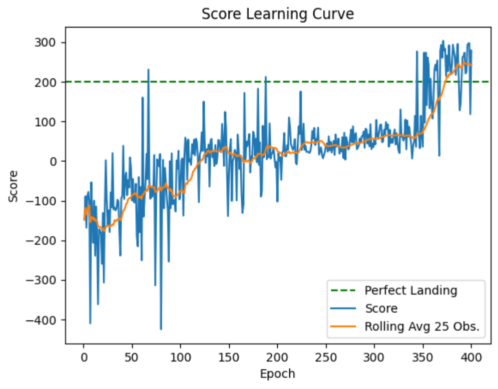


Fig. 3: Activation Functions

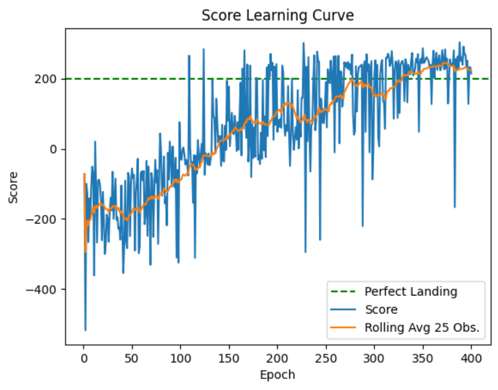
In addition to ReLU and Tanh, several other activation functions were tested, such as Sigmoid and Leaky ReLU. The Sigmoid function is primarily used for classification tasks and was inadequate for this particular application and failed to successfully place the lander on the surface. In contrast, Leaky ReLU offered comparable results to ReLU and did not improve the model's performance, and its implementation is omitted from this analysis.

VIII. DQN Results

*1) **Activation Function Results**:* Both activation functions proved capable of safely placing the lunar lander on the surface and achieving a score of 200. Individual simulation runs for each activation type are shown below in Figure 4. Both techniques require approximately 400 epochs to achieve an acceptable score of 200.



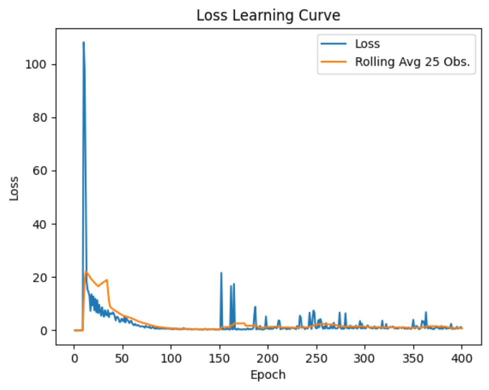
(a) ReLU Learning Result



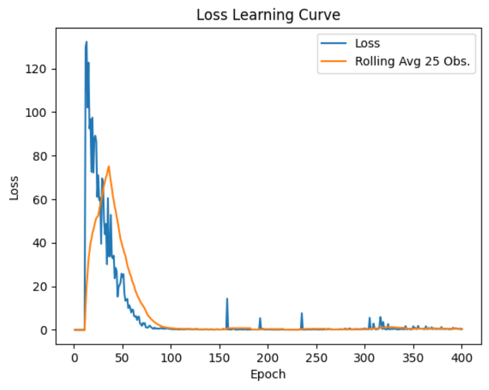
(b) Tanh Learning Result

Fig. 4: Activation Function Learning Curve with MSE Loss

An examination of the corresponding loss curves generated from both techniques in Figure 5 indicates the ReLU function minimises losses faster versus Tanh, however, the ReLU function's immediate loss minimization does not lead to steady continuous learning. In contrast, the Tanh activation function minimizes losses at a leisured pace over time but may result in more consistent learning as it reduces the realized loss volatility in later epochs versus ReLU. This subtle difference became more prevalent as both methods were exposed to repeated simulation.



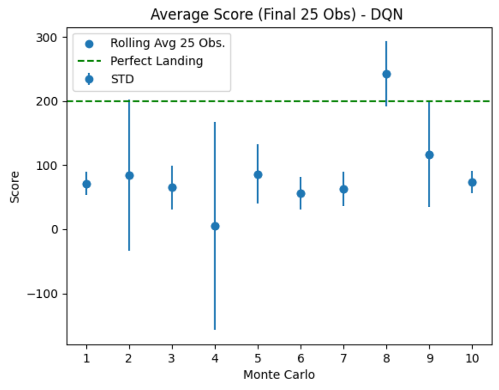
(a) ReLU Loss Result



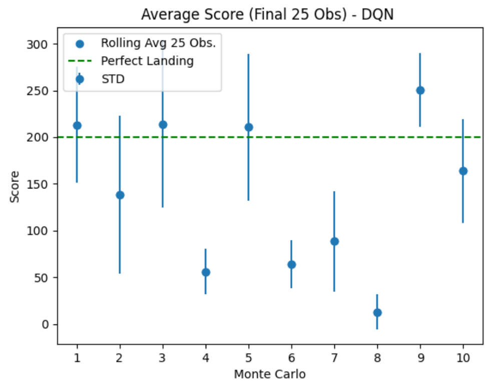
(b) Tanh Loss Result

Fig. 5: Activation Function Loss Curve with MSE Loss

To down select between options, a Monte Carlo analysis was conducted to assess model stability over multiple trials. Each technique was simulated 10 times with 400 epochs, providing insight into individual activation function's terminal composite landing score. Terminal composite landing score for each simulation was calculated by averaging the final 25 scores produced for each training epoch with results plotted below in Figure 6. An analysis of these plots indicate that the Tanh activation function was able to consistently provide a higher landing score on average versus ReLU. The Tanh activation function produced 4 terminal average values of above 200 points versus ReLU generating only 1 score above 200 points. This difference indicates the Tanh model may produce more stable results over time versus ReLU. To assess computational efficiency, the elapsed time for each simulation run was logged to create an average computational time for each technique. The Tanh function produced a slightly lower elapsed time per simulation averaging approximately 20 minutes and 57 seconds per simulation versus 22 minutes and 18 seconds for ReLU, making Tanh approximately 6% faster versus ReLU. The higher scores generated by Tanh paired with its slight advantage of computational speed led to it being selected as the activation function of preference for this model. It should be noted, the improved computational time of Tanh versus ReLU was unexpected as Tanh relies on exponentiation in it's gradient calculation. Exponentiation typically corresponds to increased computational time. One explanation for this is the lander may have learned to minimize "hovering" while utilizing Tanh, resulting in more aggressive landing approaches and shorter individual epochs, translating to faster simulation times.



(a) ReLU Monte Carlo Result



(b) Tanh Monte Carlo Result

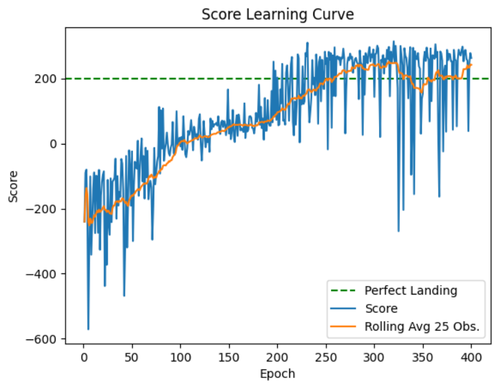
Fig. 6: Activation Function Monte Carlo Comparison

2) ***Loss Functions:*** Several loss functions were contrasted to improve on the down selected Tanh model. The loss functions tested included Mean Square Error (MSE), Mean Squared Absolute Error (MSAE), SmoothL1Loss and Huber Loss. For brevity, only two are discussed in depth in this section, MSE and Huber Loss, highlighted in Equation 6 and Equation 7. MSE is the most common loss function, taking an average of the squared error of the model's prediction versus ground truth. MSE was utilized for both previous Monte Carlo scenarios discussed above. The disadvantage of MSE is if the model makes an abnormally bad prediction of the best lunar lander next action, the error is magnified due to the squaring requirement in its formulation. Multiple bad predictions throughout a training run can significantly impact model results with MSE in this scenario. Huber loss addresses the squared error problem by combining MSE and the Mean Absolute Error (MAE) into a piecewise function. For loss differences less than a δ threshold, MSE is applied. For outlier losses, mean absolute error is applied, reducing outlier impact on model results [4]. A range of δ thresholds were analyzed with a value of 0.8 producing the highest performance results.

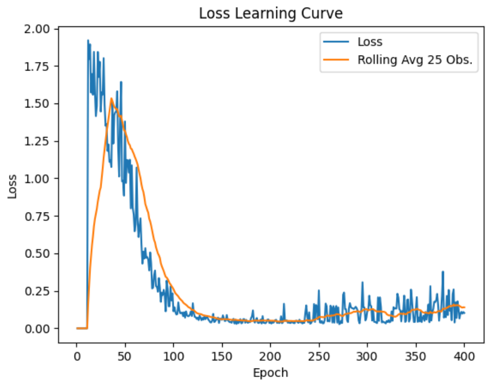
$$\text{MSE} = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{y}_i)^2 \tag{6}$$

$$\text{Huber} = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} \quad |y - f(x)| \le \delta \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{7}$$

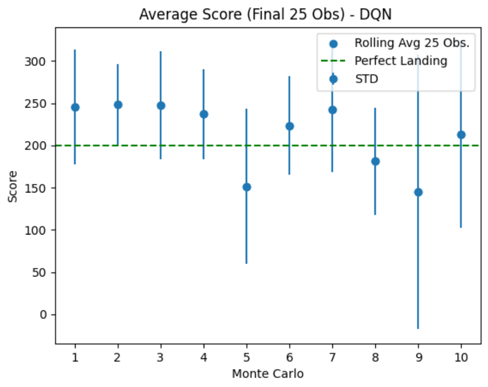
Located below are randomly selected training results when the Huber Loss technique is applied along with a Monte Carlo analysis of 10 simulations. In contrast to the MSE results shown in Figure 4 and Figure 5, the Huber Loss function was able to achieve a score of 200 at approximately 250 epochs versus MSE reaching a score of 200 at approximately 325 epochs. An examination of the loss curves for MSE and Huber Loss indicate Huber Loss learns slightly slower versus MSE with a loss convergence occurring at 250 epochs versus 75 epochs for MSE. This slower training may result in a more viable end result with respect to final model scores. The Huber Loss technique generated 7 out of 10 terminal value scores of over 200 points, with many scores approaching 250 points. MSE, in contrast, produced only 3 terminal scores with values above 200. The three terminal scores which produced values below 200 points for Huber Loss were anchored around 150 points versus MSE which produced a wide range of scores between zero and 140 points. Wall clock time was also analyzed as a performance metric. MSE produced a wall clock time of 20 minutes and 57 seconds versus Huber Loss generating a wall clock time of 18 minutes 35 seconds, making it 11.2 % faster versus it's MSE counterpart. The higher terminal value scores paired with less deviation and faster computational speed make the Huber Loss with a δ value of 0.8 a superior alternative versus MSE. Huber Loss was implemented in all further lunar lander optimizations.



(a) Huber Loss Simulation Results - Score



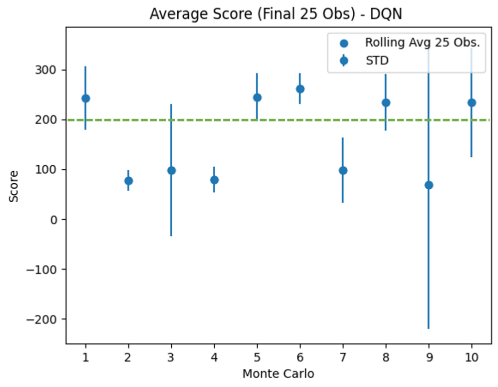
(b) Huber Loss Simulation Results - Loss



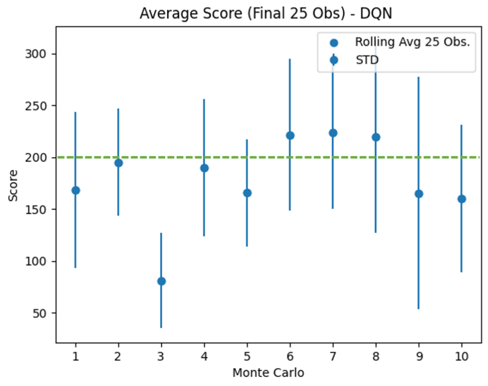
(c) Huber Loss Monte Carlo Results

Fig. 7: Huber Learning Analysis

3) ***Replay Buffer Size:*** To continue model optimization, a scenario analysis was performed to determine if the default training batch size of 1,000 observations could improve either the lunar lander's end performance or improve computational speed. To flex this parameter, the default batch size of 1,000 observations was halved to 500. The replay buffer was then doubled to 2,000 observations and the results were contrasted to determine if any improvement had occurred. A reduced batch size of 500 observations produced less standard deviation versus the other two alternatives, however, it only produced 5 terminal value scores of above 200 points. In contrast the increased batch size of 2,000 only produced 3 terminal scores of above 200 points and introduced significant variation in maximum and minimum scores experienced throughout each simulation as seen in Figure 8. The wall clock time produced by the smaller batch size of 500 was 16 minutes and 30 seconds whereas the larger batch size of 2,000 required 28 minutes and 54 seconds to run. In contrast to these scenarios, the batch size of 1,000 performed as expected, residing within these upper and lower wall clock time bounds at a time of 18 minutes and 35 seconds. The improved wall clock time produced by the smaller batch size of 500 does not out weigh the decrease in performance results generated by the smaller replay buffer. As a result, the default batch size of 1,000 observations was retained.



(a) Batch: 500 Monte Carlo Results



(b) Batch: 2,000 Monte Carlo Results

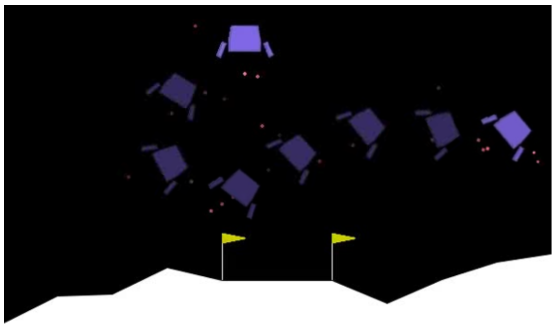
Fig. 8: Batch Size Analysis

4) ***Optimized DQN Structure:*** After multiple iterations, the final DQN structure offering the most consistent and reproducible results is defined below:

- A fully connected neural network
- 3 hidden layers of 200 neurons each
- **Activation Function:** Tanh
- **Optimizer Type:** Adam
- **Loss Function:** Huber Loss $\delta = 0.8$
- **Replay Buffer Size:**1,000

- **Learning Rate:** 0.001
- **Epochs** = 400

The neural network defined above was able to consistently produce results similar to Figure 9a and Figure 9b, below. Figure 9a shows a time lapse of the initial first exploration epoch of the lunar lander. The neural network is in its most raw form in terms of learning during this stage. The lander follows a random path as the agent begins to explore its environment, collect experiences and learn from its previous actions. Figure 9b below highlights a typical landing performed by the agent once the neural network has been optimized.


(a) Epoch = 1

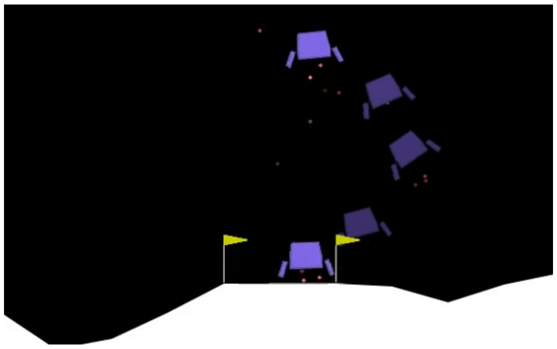

(b) Epoch = 400

Fig. 9: Untrained (top) and Trained (bottom) Lunar Lander

IX. Conclusion

Based on the Discrete Q-Learning and DQN results described above, it is concluded Q-learning alone is insufficient to successfully place the lander on the surface of the Moon. The information lost during discretization of the environment leaves gaps of state space knowledge in the agent's memory, thus hindering its end performance. This, in essence, would be the equivalent of attempting to drive a car at highway speeds based off of photographs taken every half second. In contrast, the DQN eliminates the need for discretization and vastly improves the agent's memory and decision making capabilities. The Q-table of cataloged experiences created by the lander in Q-learning is transformed into an intricate neuron structure capable of storing more complex combinations of information. This memory structure provides a foundation for agent learning and allows complex relationships between the lander's current state and thrusters to be uncovered and improved upon, resulting in more successful landing attempts. The Discrete Q network proved it was capable of improving over time, however, it's capabilities plateaued at a score of approximately -50 points. In contrast, the DQN proved to be a more capable learner and was more successful in recreating high scores over time as evidenced by Monte Carlo analysis. The malleability of the neural network facilitates significant flexibility in hyperparameter tuning. Changes in individual hyperparameters produced unique end results in lunar lander behavior. In this particular scenario, the Tanh activation offered superior performance in terms of terminal landing scores, producing 4 out of 10 final scores of approximately 210 points each versus the more commonly used ReLU activation function, which only produced 1 out of 10 scores above 200 points. Tanh also promoted a more aggressive initial approach with respect to lunar lander flight path characteristics, resulting in a faster wall clock times versus ReLU. This end result was surprising, considering ReLU is an optimized version of Tanh and removes the need for exponentiation in its calculations. This would typically promote improved calculation times during training.

Loss function tuning was another area of the neural network offering significant results when tuned. In this particular environment, the standard MSE loss function proved to be very sensitive to the large negative scores generated by the lander during either the initial pure exploration phase or the persistent exploration phase included in the epsilon greedy strategy. These large negative outliers were magnified due to the MSE loss function's requirement of squaring the estimated and observed differences. A Huber Loss technique was implemented to provide a piecewise loss function capability, possessing the ability to switch between an MSE loss function and an MAE function when a δ threshold was surpassed. After tuning, it was determined a δ value of 0.8 provided the best results. Furthermore, upon implementing the Huber Loss technique, the lunar lander's loss learning curve decayed to zero much slower versus the pure MSE method. However, this slower learning rate produced more consistent terminal landing scores. The Huber Loss technique produced 7 out of 10 terminal landing scores above the 200 point threshold versus 4 for the Tanh activation function paired with MSE alone. Huber loss provided more consistent scoring with less standard deviation versus MSE and also generated significantly higher average end values. The average terminal value score above the 200 point threshold for MSE was 223.7 versus 237.14, providing a 6% improvement in average score along with 1.3X more success in terms of achieving terminal landing scores of above 200.

To continue model development, replay buffer size was also explored under the assumption that a smaller batch size of lander training experiences would result in faster wall clock speed while preserving scoring results. Additionally, an increased batch size was explored to determine if an increased training memory bank could significantly improve the lander's final scores with minimal effect on wall clock time. The Monte Carlo simulations produced for both the reduced batch size of 500 and increased batch size of 2,000 resulted in decreased lander performance. The reduced batch size of 500 correlated to an 11% improvement in wall clock time versus the default batch size of 1,000 experiences, however, this reduction in batch size introduced significantly worse terminal performance in the lander's final scores, producing only 5 scores above 200 points versus the default batch size succeeding 7 times. The increased batch size of 2,000 also offered inferior results versus the default value of 1,000. The larger replay experience bank produced only 3 terminal value scores of above 200 points. Additionally, this decreased performance was coupled with a 55.5% increase computation elapsed time, making it the worst option of the 3. After exploration it was determined the default batch size of 1,000 offered the most consistent performance for only slightly more computational time versus the smaller replay memory option.

It should be noted there are many ways to successfully implement and train a neural network to provide acceptable results. Significant hyperparameter exploration was required to generate the end results discussed above. Extensive scenario analysis of other neural network structures such as hidden layer number, neuron, count and dropout were also explored in addition to various activation functions such as Sigmoid, Leaky ReLU and hybrid networks. Additional optimization parameters other than ADAM were also implemented such as Gradient Descent, RMSProp, ADAgrad with no advantages becoming present in final results. Other more basic parameters were also varied such as learning rate, agent memory size, and epsilon decay. However, these changes only deteriorated agent performance and were omitted from this paper. The flexibility of neural nets and their ability to accommodate continuous state space models such as the lunar lander environment allow the Q-learning algorithm to be implemented in much more powerful and efficient ways versus it's original tabular nature. However, Q-learning provides the framework in which this model is built and reaffirms it's continued importance in many modern reinforcement learning techniques applied today.

X. Contributions and Release

- **Selmo Almeida** - Performed hyperparameter Tuning, created overleaf organization document, and contributed to final report.
- **Jason Peloquin** - Developed DQN algorithm, performed hyperparameter tuning and analysis, performed Monte Carlo analysis, and contributed to final report.
- **Jacob Reed** - Developed Discrete Q-Learning algorithm, performed Monte Carlo analysis, managed version control (GitHub development), developed virtual environment establishment, and contributed to final report.

The authors grant permission for this report to be posted publicly. The discussed reinforcement algorithm scripts with workflow and setup instructions can be found at: https://github.com/reedjacobp/LunarLanderProject.

References

[1] M. J. Kochenderfer, T. A. Wheeler, and K. H. Wray, Algorithms for decision making, 1st ed. Cambridge, MA: The MIT Press, 2022.

[2] A. Scorsoglio, R. Furfaro, R. Linares, and B. Gaudet, "Image-based deep reinforcement learning for Autonomous Lunar Landing," https://arc.aiaa.org/doi/10.2514/6.2020-1910, 10-Jan-2020. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2020-1910. [Accessed: 20-Apr-2023].

[3] G. Ciabatti, S. Daftry, and R. Capobianco, "Autonomous planetary landing via Deep Reinforcement Learning and Transfer Learning," 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 2031–2038, Jun. 2021.

[4] V. Yathish, "Loss functions and their use in neural networks," Medium, 04-Aug-2022. [Online]. Available: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9. [Accessed: 25-Apr-2023].

[5] S. Sharma, "Activation functions in neural networks," Medium, 20-Nov-2022. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed: 27-Apr-2023].

[6] S. Potter, "NASA, Northrop Grumman Finalize Moon Outpost Living Quarters Contract," NASA, 08-Jul-2021. [Online]. Available: https://www.nasa.gov/press-release/nasa-northrop-grumman-finalize-moon-outpost-living-quarters-contract. [Accessed: 29-Apr-2023].

[7] O. Klimov, "Lunar Lander," https://gymnasium.farama.org/environments/box2d/lunar_lander/