Lunar Lander Problem Using Reinforcement Learning

Divya Arunraj, Punya Prateek, Pranay Vaddepalli, Hemanth Komanahalli Ramu, Vipan Koul, Gokulan Nithianandam
Department of Computer Science
University of Bath
Bath, BA2 7AY

1 Problem Definition

Our objective is to use Reinforcement Learning(RL) to solve the 'Lunar Landing' problem available in OpenAI's gym toolkit. RL is the branch of machine learning that follows computational approach to goal-directed learning based on interaction Sutton and Barto (2018). A well-defined physics engine is utilised to simulate a scenario in which the lander must land in a precise area under low gravitational conditions. The basic goal of the game is to direct the agent to the landing pad as efficiently as possible. The state space is continuous, much like in real physics, whereas the action space is discrete. In this project, we will train a Deep Q-Network (DQN) agent to solve the "Lunar Landing" problem Verma (2021).

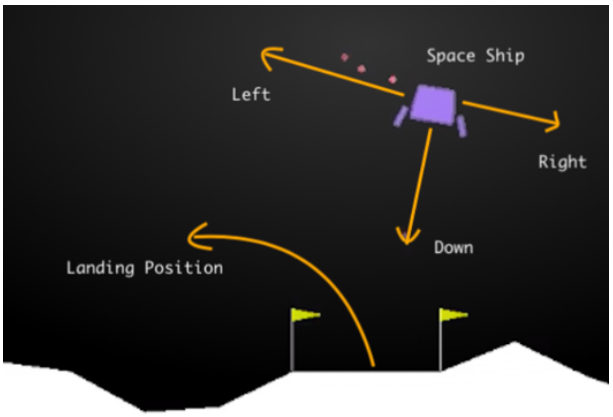

Figure 1: Lunar Lander

States: The state/observation is the current state of the environment. There is an 8-dimensional continuous state space and a discrete action space.

1. Horizontal Position
2. Vertical Position
3. Horizontal Velocity
4. Vertical Velocity
5. Angle
6. Angular Velocity
7. Left Leg Contact
8. Right Leg Contact

Actions: For each state of the environment, the agent performs an action depending on the current state. The agent can select one of the four distinct actions: 'Do Nothing', 'Fire Left Engine', 'Fire Right Engine', or 'Fire Main Engine'. Because we are firing both left and right engines, torque is introduced on the lander, causing it to twist and making stabilisation harder OpenAI.

Reward Functions: The reward for moving from the top of the screen and landing on the landing pad with zero speed ranges between 100 and 140 points. Each leg contact made with the ground earns a reward of 10 points. In each frame, firing the primary engine results in -0.3 reward points and firing the side engine results in -0.03 point rewards. If the lander crashes or comes to rest, an additional reward of -100 or +100 points is awarded, resulting in the termination of the episode Diddigi et al..

2 Background

The common challenge faced in reinforcement learning is that we are exposing the agent to an unknown environment in which it does not know the consequences of its actions and has no awareness about its present state. The agent receives a reward for performing an activity and must make decisions depending on its actions. As a result, it is difficult for the agent to first get acquainted with the environment and then optimize its policies. Thus, we need to use some pre-existing algorithms to help the agent reach its goal. In order to solve the lunar landing problem, we can explore two algorithms: Deep Q-Network and Double Deep Q-Network Continuous Control with Deep Reinforcement Learning.

Comparing how the two algorithms fare against each other:

1. **Deep Q-Network (DQN):**
    - Q-Learning, but with Deep Neural Network for function approximation.
    - Q-Learning fails because of overestimation of action values. These over-estimations result from a positive bias introduced by using the maximum expected action value for approximation.
    - In reinforcement learning, both the input and the target change constantly during the process and make training unstable.

    DQN overcomes unstable learning by mainly 4 techniques:

    - Experience Replay
    - Target Network
    - Clipping Rewards
    - Skipping Frames
2. **Double Deep Q-Network (DDQN):**
    - We apply the double estimator to Deep Q-learning to construct Double Deep Q-learning (Double DQN), a new off- policy reinforcement learning algorithm.
    - DDQN uses two separate Q-value estimators, each of which is used to update the other. Using these independent estimators, we can get unbiased Q-value estimates of the actions selected using the opposite estimator. We can thus avoid maximization bias by disentangling our updates from the biased estimates.

2.1 Existing scientific literature

There already exists a literature work Gadgil, Xin and Xu (2020) on solving the lunar landing problem. The research study investigated how well Sarsa and DQN agents performed in the presence of increased ambiguity. Sarsa algorithm performed substantially worse with noise observation. This disadvantage is addressed by Partially Observable Markov Decision Process(POMDP), which uses belief vectors to describe a distribution across probable next states. However, the DQN agents outperformed the Sarsa and POMDP agents.

3 Method

To solve the lunar landing problem, we are going to implement DQN algorithm. DQN stands for Deep Q-Network. To compute optimum actions, this technique employs deep neural networks. Our purpose of this study is to investigate the effect of the number of neural network layers on algorithm's performance.
Explanation of DQN algorithm:

1. Initialize replay memory capacity.
2. Initialize the network with random weights.
3. For each episode:
   (a) Initialize the starting state.
   (b) For each time step:
      i. Select an action.
         A. Via exploration and exploitation.
      ii. Execute selected action in an emulator.
      iii. Observe reward and next state.
      iv. Store experience in replay memory.
      v. Sample random batch from replay memory.
      vi. Pre-process states from batch.
      vii. Pass batch of pre-processed states to policy network.
      viii. Calculate loss between output Q-values and target Q-values.
         A. Requires a second pass to the network for the next state.
      ix. Gradient descent updates weights in the policy network to minimize loss.

3.1 Why we chose DQN method?

After reviewing research publications, we discovered that DQN agent with a neural network layer of hidden neurons consistently achieves greater average rewards. Furthermore, Q-learning is an off-policy algorithm that learns the optimum policy using the absolute greedy policy by picking the next action that maximises the Q-value. Because this is a simulation, the agent's performance during the training phase is irrelevant. In such cases, the DQN agent would outperform since it learns an optimum policy that we finally switch to.

4 Results

- The agent's learning curve for various learning rate of Policy Neural Network and Target Neural Network is represented below in the figures.

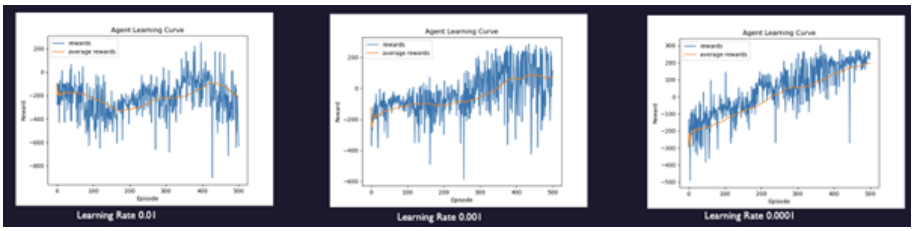

Figure 2: Agent learning curve for learning rates 0.01, 0.001 and 0.0001

- The agent's learning curve for various batch size of the replay memory is represented below in the figures.

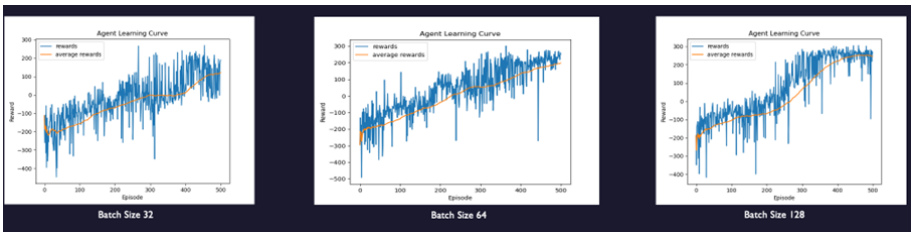

Figure 3: Agent learning curve for batch size 32, 64 and 128

- The agent's learning curve for various Target Neural Network weight update interval is represented below in the figures.

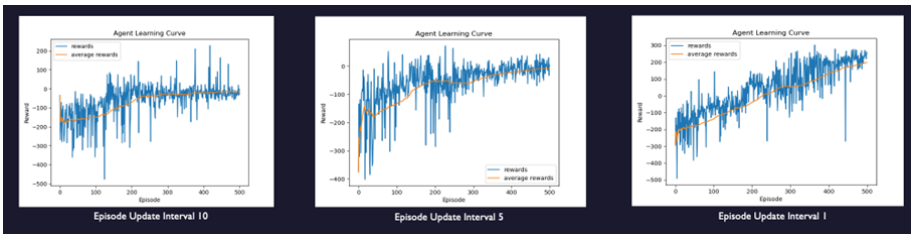

Figure 4: Agent learning curve for Target Neural Network weight update interval 10,5 and 1

- The agent's learning curve for various discount factor is represented below in the figures.

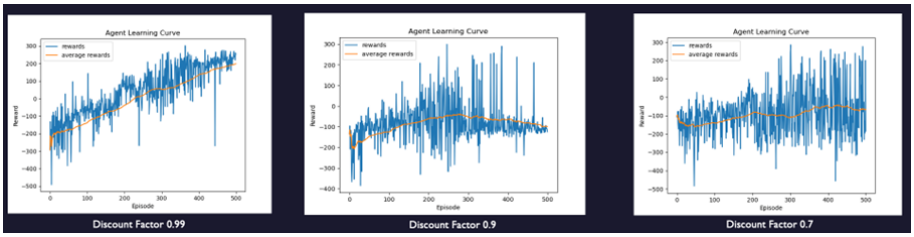

Figure 5: Agent learning curve for discount factor 0.99, 0.9 and 0.7

- The agent's learning curve for various Epsilon decay rate is represented below in the figures.

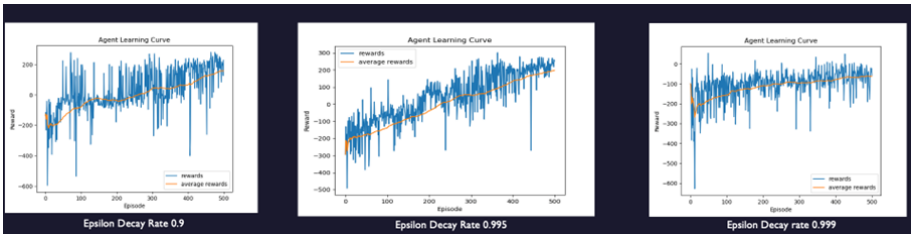

Figure 6: Agent learning curve for Epsilon decay rate 0.9, 0.995 and 0.999

- The agent's learning curve for the overall tuned hyper-parameter's is shown below.

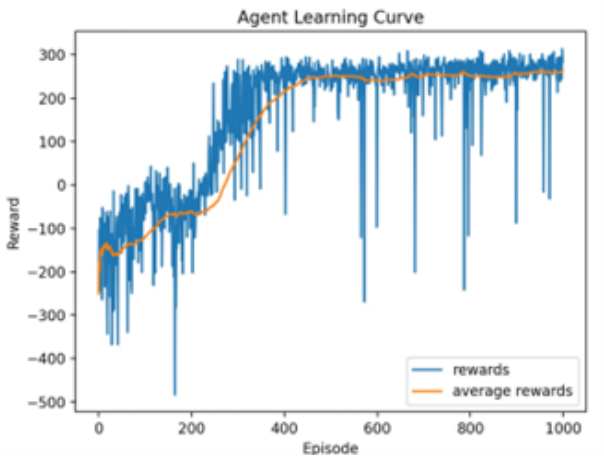

Figure 7: Overall tuned hyper-parameter Learning Curve Graph

5 Discussion

The figure shows the learning curve after the process of fine tuning with the benchmark hyper parameters, the agent's average reward remained constant after 400 episodes as the agent attained the optimal policy. The effect of hyper-parameter's on the agent's learning process is discussed below.

5.1 Policy network and target network learning rate

The Deep Neural Network accepts new values compared to the old value if the learning rate is set between 0 and 1. This value is added to the current Q value, causing it to move in the direction of the most recent update. When the learning rate is set to 0, the values are never updated, and the Q value remains the same as it was at the start. When the learning rate is set to 0.9, the old values are updated quickly. We trained the model using learning rates of 0.001, 0.0001, and 0.01 to evaluate it with different learning rates. The learning 0.0001 lets the agent to find the optimal policy in fewer episodes, as seen in the figure. The reward values diverge in a large proportion with a greater learning rate, limiting the agent's ability to find the best policy in fewer episodes. At a slower learning rate, reward values diverge in a smaller proportion, allowing the agent to find the best policy in fewer episodes.

5.2 Replay memory batch size

To test the agent training process, different batch sizes are used. The number of random samples collected from the replay memory to train the policy neural network is referred to as batch size. The batch sizes used to test the agent are 32, 64, and 128. The agent's learning curve for various batch size is shown in the figure. According to the evaluation, the agent trained with batch size 64 and 128 receives a positive reward after around 300 episodes, as opposed to batch sizes 32 where the agent receives positive rewards before 300 episodes. The rewards convergence is worse in batch sizes of 32 because small batch sizes require a lot of iteration to converge. The agent received a steady reward for each episode with a batch size of 64 and 128, which allows for improved convergence. Furthermore, when compared to agents with batch sizes of 64 and 128, the agent with batch size 128 received the highest average reward. Finally, we observed that 128 is the best batch size for training the agent.

5.3 Update interval of the target neural network weights

With episode intervals of 1, 5, and 10, we updated the target network weights. To bring the output q values closer to the target Q value, different update intervals are trailed. We thought that updating the target weights on every 10 episodes would give greater convergence, however the process had

a major impact on the agent's performance, as seen in the figure, where the average rewards were negative during a 500-episode period. We determine that a larger interval is detrimental to the agent's performance. When compared to a 10-episode interval, we lowered the interval to 5 episodes and the agent's performance improved. We decided that a short interval provides higher performance based on the aforementioned observations, and we trained the agent while updating the target network weight for each episode, and the agent's performance was superior, as shown in the figure.

5.4 Discount factor

The discount factor determines the pace at which rewards in the distant future are accepted in comparison to rewards in the near future. If the discount factor is set to zero, the agent can only learn the actions with instant rewards. The agent can only learn actions based on the total of all of its future rewards when the discount factor is set to one. While training the agent, the discount factor of 0.99, 0.9, and 0.7 was used, and the learning curve for the agent with discount factor is shown in the figure. In comparison to the discount factors of 0.9 and 0.7, the agent trained with the larger discount factor of 0.99 offered the biggest reward at the end of 500 episodes. Furthermore, when the agent is trained with a discount factor of 0.99, the reward convergence is better.

5.5 Epsilon decay rate

For a given state and action pair, Epsilon expresses the policy randomness. In general, the epsilon value decays as the number of episodes increases. During training, the agent switches from choosing a randomised policy to deterministic policy over a period of time. The epsilon decay parameter specifies the time after which the agent begins to choose the deterministic policy. We trained the agent with epsilon decay of 0.999, 0.995, and 0.9 for this paper, and the learning curve of the agent is shown in the figure. When epsilon decay is 0.9, the agent performs better than with epsilon decay as 0.999 or 0.995. At the epsilon decay of 0.9, we believe the agent established a better balance between exploration and exploitation approach.

6 Future Work

In future work, we would have explored to implement DDQN algorithm on the same Lunar Lander environment. From what we understand, the difference between DQN and DDQN is in the calculation of the target Q-values of the next states. By using Double DQN algorithm, we can check how the agent would have performed for the same environment, as this would have given a more holistic overview of the robustness of the agent.

7 Personal Experience

It was tough to include the replay memory into the programming. Going with the approach of tweaking hyperparameters allowed the authors to have a strong understanding of the impacts of the aforementioned hyperparameters in the DQN agent.

References

Continuous control with deep reinforcement learning. Available from: https://cse.buffalo.edu/~avereshc/cse_demo_day_fall19/Vishva_Nitin_Patel_Leena_Manohar_Patil_poster.pdf.

Diddigi, R.B., Tasse, G.N., Vidal, Y., Krishnagopal, S. and Rajaee, S. Neuromatch academy: Deep learning. Available from: https://deeplearning.neuromatch.io/projects/ReinforcementLearning/lunar_lander.html.

Gadgil, S., Xin, Y. and Xu, C., 2020. Solving the lunar lander problem under uncertainty using reinforcement learning. *2020 southeastcon*. IEEE, vol. 2, pp.1–8.

OpenAI. A toolkit for developing and comparing reinforcement learning algorithms. Available from: https://gym.openai.com/envs/LunarLander-v2/.

Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. MIT press.

Verma, S., 2021. Train your lunar-lander: Reinforcement learning. Available from: https://shiva-verma.medium.com/solving-lunar-lander-openaigym-reinforcement-learning-785675066197.

8 Appendices

8.1 Appendix A: Problem Description

Our goal is to solve OpenAI's lunar landing environment gym kit with reinforcement learning method. RL is computational approach to goal-directed learning from interaction [1]. The environment simulates the situation that the lander needs to land in a specific location under low gravity conditions and implemented a well-defined physics engine. The main purpose of the game is to direct the agent to the landing pad that is as softly and fuel efficient as possible. The state space is continuous like real physics, the action space is discrete. In this project, we will solve the "Lunar Landing" problem by training a Deep Q-Network (DQN) agent.

8.2 Appendix B: Learning Rate Experiments

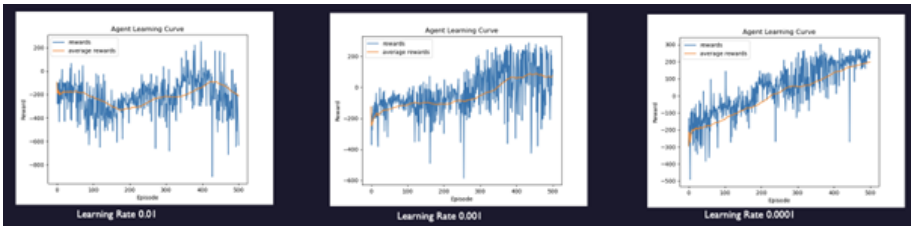

Figure 8: Agent learning curve for learning rates 0.01, 0.001 and 0.0001

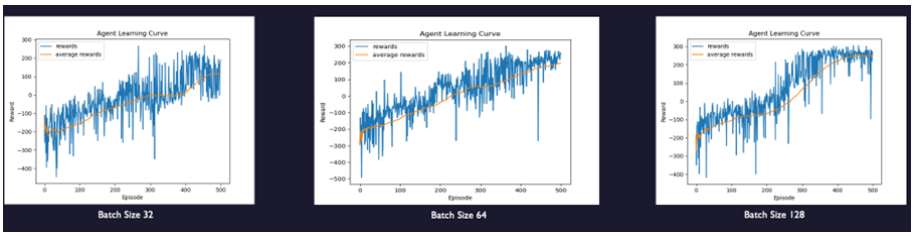

Figure 9: Agent learning curve for batch size 32, 64 and 128

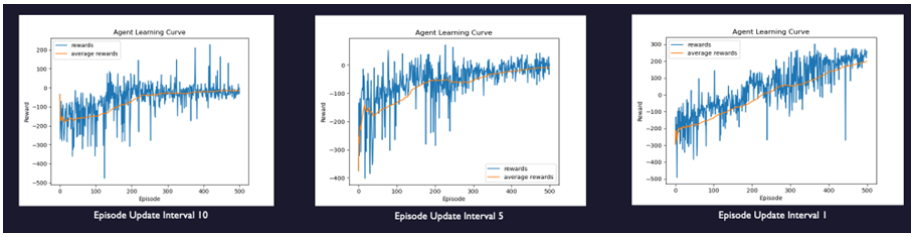

Figure 10: Agent learning curve for Target Neural Network weight update interval 10,5 and 1

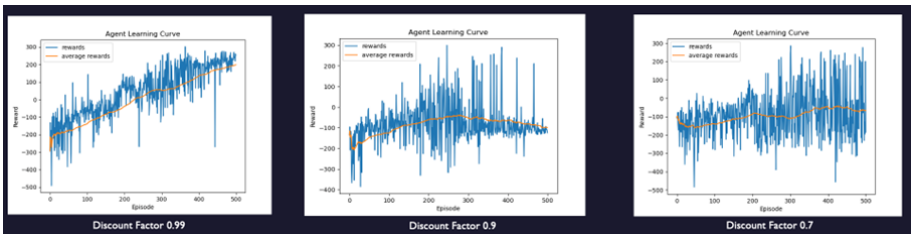

Figure 11: Agent learning curve for discount factor 0.99, 0.9 and 0.7

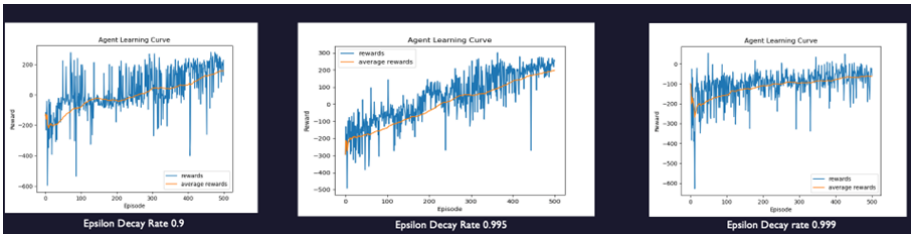

Figure 12: Agent learning curve for Epsilon decay rate 0.9, 0.995 and 0.999

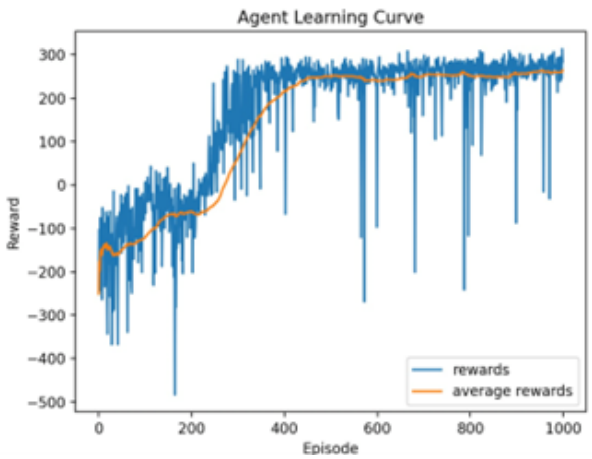

Figure 13: Overall tuned hyper-parameter Learning Curve Graph