Solving the Lunar Lander Problem: Epsilon-Greedy in Sarsa and Deep Q-Learning

Julia Vaghy **Joris Peters**

Abstract

The present study is aimed at solving OpenAI's Lunar Lander problem using reinforcement learning algorithms Sarsa algorithm and Deep Q-Learning. Their performances are compared, and several state space discretization strategies and the effects of different ϵ-greedy schedules are explored. The results suggest that, while adopting a more refined discretization strategy and ϵ-greedy schedule affects the performance of SARSA agents, no visible difference in performance was found for the Deep Q-Learning agents when modifying the ϵ-greedy schedule. Only the Deep Q-Learning algorithm converged successfully, suggesting that Deep Q-Networks are better suited for the complexity of the Lunar Lander problem's state space.

1 Introduction

1.1 Lunar Lander

The present study aims to solve OpenAI's LunarLander-v2 problem using reinforcement learning. The task is one of the Box2D environments from the OpenAI gym library (Brockman et al., 2016). The Lunar Lander problem is set in a 2-dimensional lunar scene under low-gravity conditions and requires a spacecraft agent to land on a landing pad centered around the coordinates (0,0) as shown in Figure 1, while using a minimal amount of fuel. This version of the problem has discrete actions and continuous state space.

Action space There are the following four possible actions $a \in A$ (with action space A) of which only one may be taken at a time step.

1. Do nothing
2. Fire left orientation engine
3. Fire main engine
4. Fire right orientation engine

State space The state space S consists of 8 variables - 6 continuous and 2 discrete (boolean):

1. X distance from target site
2. Y distance from target site
3. X velocity
4. Y velocity
5. Angle of ship
6. Angular velocity of ship
7. Left leg is on the ground (boolean)
8. Right leg is on the ground (boolean)

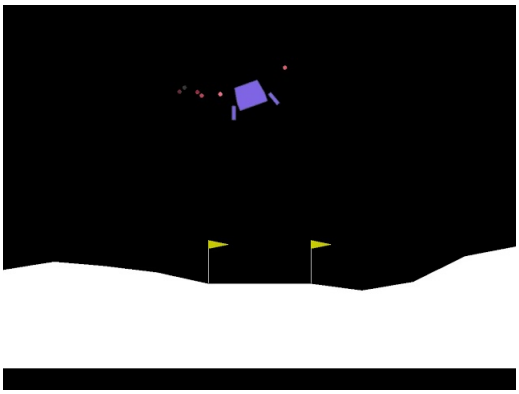

Figure 1: The Lunar Lander problem

Episodes An episode finishes when the lander crashes or comes to rest on the lunar surface. The episode is only considered successful if the lander comes to rest at the landing pad.

Reward The agent receives a reward r_t at each time step t of an episode based on the following conditions:

- Move from the top of the screen to the landing pad and zero speed: +100-140 points (the agent loses this reward if it leaves the landing pad)
- Firing main engine: -0.3 points
- Lander crashes: -100 points
- Lander comes to rest: +100 points
- Ground contact for each leg: +10 points
- Lander comes to rest at the landing site: +200 points

2 Literature review

Yu (2019) solved OpenAI's Lunar Lander problem using a Deep Q-Network (DQN). The paper has several suggestions for training a successful DQN agent. Firstly, since the most significant reward comes at the end of episodes from attaining successful landing, a near-one temporal discount factor γ is preferred. Second, added exploration is necessary in order to properly explore the state space, and the exploration parameter ϵ should decrease over time for smooth convergence. Finally, a Q-network with a large hidden layer leads to better results than a deep network.

Other work (Gadgil, Xin, & Xu, 2020) describes several discretization and exploration methods in an attempt to apply the Sarsa learning method to this continuous state space problem. Notably, they suggested the discretization scheme we adopt in Discretization strategies 2.1 and 2.2, as explained in Section 3.1.2.

Xu et al. (2018) also highlight the importance of exploration, however, the authors used a different method to combine Deep Q-Networks and exploration, namely, the Deep Sarsa and Q Networks (DSQN) algorithm. This entails using exploration while calculating the temporal difference target y_i instead of a greedy strategy, as explained in section 3.2. In contrast, Yu (2019) incorporated exploration in the agent's interactions with the environment instead of during weight updates.

The current study aims at extending this work. Particularly, this is an attempt to (a) solve this problem with the Sarsa and Deep Q-learning algorithms and (b) to explore the effects of different ϵ-greedy schedules on the performance of these algorithms.

3 Methodology and Results

Two algorithms, Sarsa and Deep Q-learning, were compared. Performance was measured based on the score (cumulative reward over all time steps) for each episode. We define success as the algorithm converging to a score of 200+ per episode. For both algorithms, the goal is to learn the state-action values $Q(s, a)$ for each state-action combination. The state-action value function denotes the expected summed discounted rewards resulting from taking action $a \in A$ in state $s \in S$ under policy π:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\},$$

where s_t and a_t are the state and action taken at time t, r_n is the reward at time step n, and γ is the temporal discount factor.

3.1 Sarsa

3.1.1 The algorithm

For Sarsa, state-action values are stored in a Q-table of size $|S \times A|$, and are updated at each time step t using the following update rule:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)],$$

where s_t is the state at time t, a_t is the action the agent took, a_{t+1} the action taken from state s_{t+1} according to the policy π, α the learning rate and γ the temporal discount factor.

Given these values, an action can be selected in a state, according to the ϵ-greedy policy (defined below) with different values of ϵ as a function of the episode number to encourage early exploration. At any time step t, the next action is chosen by sampling the Q-value-maximizing action with a probability of $1 - \epsilon$ and one of the other actions with a probability of ϵ. This gives the following equation for the policy $\pi_t(a)$ from which the actions are sampled:

$$\pi_t(a) = \begin{cases} (1 - \epsilon) + \epsilon/|A| & \text{if } a = \text{argmax}_{a \in A} Q_t(a) \\ \epsilon/|A| & \text{otherwise} \end{cases}$$

3.1.2 Discretization and optimizing exploration

The Sarsa algorithm requires a discrete state and action space. Thus, it was necessary to translate the default continuous state space to a discrete one. Due to the exponential increase in state-action Q-values with the number of discretization bins - $|\mathcal{D}_Q| \sim |S_c|^N$, with the domain $\mathcal{D}_Q$ of Q-values, the number of discretization bins N and the subspace of continuous states $S_c \subset S$ - only a selected amount of discretization intervals is computationally feasible.

Discretization strategy 1 As the minima and maxima of the 6 continuous state variables were not defined in the documentation, we ran 10000 experiments with a random agent to approximate these values. Then, each variable was discretized over $N = 5$ and, then, $N = 10$ equal intervals in their respective ranges, setting the lower and upper bounds to negative and positive infinity, respectively. This, however, this resulted in Q-tables with as many as, respectively, $5^6 * 2 * 2 * 4 = 250000$ and $10^6 * 2 * 2 * 4 = 16000000$ entries. For these discretization strategies, no significant improvement in performance was found on 10000 training episodes.

Discretization strategy 2.1 Hence, the state space was further reduced. Since the agent's actions directionally influence its horizontal, vertical, and angular velocity, it may be sufficient for the model to represent whether the 6 continuous state variables have positive or negative values, as suggested by Gadgil et al. (2020). We restructured discretization to N=3 intervals, the middle interval being a small range around zero. The middle interval was set to $0 \pm 5\%$ of the previously found range between the minima and maxima for each variable.

Experiment 2.1 Several smaller experiments were run on 2000-10000 episodes n to observe how different amounts of exploration affect learning. Each experiment was averaged over 3 runs. The exploration was controlled by the greedy parameter ϵ which we optimized by hand over several intervals of the episodes.

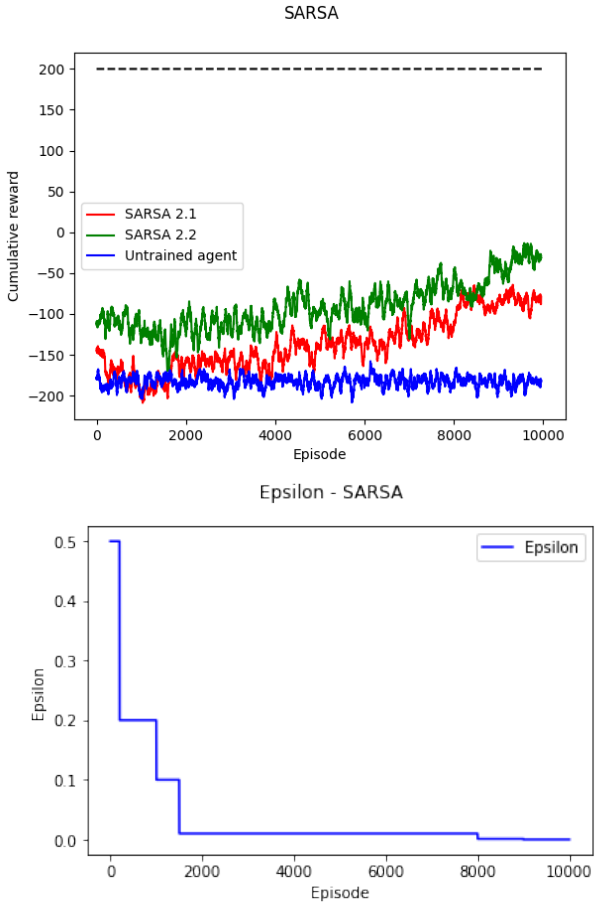

Figure 2: The top figure shows cumulative reward per episode over 10000 training episodes using Sarsa with Discretization strategy 2.1 (red) and 2.2 (green), averaged over 5 experiments. In addition, a random agent's performance (blue) is added for reference and the goal of 200 cumulative reward per episode is marked with a dashed line. The values of epsilon over the episodes are plotted in the bottom figure.

Results 2.1 It appeared that the best performance occurred with $\epsilon = 0.01$, and having 1000 final episodes with $\epsilon = 0$ increased performance further. Some early exploration with $\epsilon = 0.5$ and $\epsilon = 0.2$ lead to worse initial performance, but increased the performance the agent later converged to. On the other hand, too much initial exploration and initial exploration with $\epsilon = 1$ appeared to be counter-productive. The most optimal strategy turned out to be $\epsilon = 0.5$ for $n \in (0, 200)$, $\epsilon = 0.2$ for $n \in (200, 1000)$, $\epsilon = 0.1$ for $n \in (1000, 1500)$, $\epsilon = 0.01$ for $n \in (1500, 8000)$, $\epsilon = 0.001$ for $n \in (8000, 9000)$, and $\epsilon = 0$ for $n \in (9000, 10000)$. Nonetheless, even after optimizing epsilon, the algorithm converged to suboptimal (below scores of 200) performance around 10000 training episodes, as shown in Figure 2. Running the algorithm on 15000 episodes showed no further improvements, implying the algorithms have converged at 10000.

Discretization strategy 2.2 A weak point of strategy 2.1 may have been a too broad middle interval, leading to imprecise actions around critical values (e.g., close to the landing platform). Thus, we decreased it to 0 ± 0.05, and ran the algorithm on the same set-up as with Discretization strategy 2.1.

Results 2.2 Performance improved, however, remained suboptimal, as shown in Figure 2. Again, increasing the number of episodes resulted in no further improvement.

3.2 Deep Q-Learning

Observing the poor performance that is likely due to the complexity of the state space, Deep Q-learning offers itself as a more suitable alternative for learning the state-action values in this continuous environment. In the present implementation of Deep Q-learning, an artificial neural network was used instead of a Q-table. During training, two techniques, (1) experience replay and (2) the use of a second target network that was suggested by Lillicrap et al. (2015), helped to improve stability and performance.

Firstly, transition experiences $(s_t, a_t, r_t, s_{t+1}, d_t)$ at time steps t are stored in a replay buffer D to enable training on approximately independent identically distributed observations for better stability. d_t, the boolean *done* variable of the Gym environment which indicates whether or not the game terminated at t, was added traditional state-action-reward-state tuple to account for the termination in the network update, as described below. Due to the fixed buffer size, after collecting experiences that exceed the buffer size $|D|$ each newly added transition experience overrides the oldest one.

In each experience replay episode i, we uniformly sample a batch B of previous transition experiences $(s_t, a_t, r_t, s_{t+1}, d_t) \sim U(D)$ from the buffer. The Q-network is then trained by minimizing the loss functions $L_i(\theta_i)$ on this batch:

$$L_i(\theta_i) = \frac{1}{2}\,\mathbb{E}_{(s_t,a_t,r_t,s_{t+1},d_t)\sim U(D)}[(y_i - Q(s_t, a_t; \theta_i))^2],$$

where the temporal difference target equals

$$y_i = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_i^-).$$

Here, the second improvement is made, by using a second target network to approximate the target values $Q(s_{t+1}, a_{t+1}; \theta_i^-)$ instead of the trained main network with weights θ_i^-. The weights θ_i^- of the target network are a delayed version of the main network weights θ_i after training on the previous experience replay episode i. This avoids catastrophic forgetting which may otherwise occur, since weight updates of one state input may affect the network's responses to other inputs. At each i, the weights are linearly interpolated according to time constant τ ($\tau = 0.01$ in this experiment):

$$\theta_{i+1}^- = (1 - \tau)\theta_i^- + \tau\theta_i$$

Now that all components of the loss are in place, gradient descent is performed on the weights of the main network according to:

$$\theta_{i+1} = \theta_i - \alpha \nabla_{\theta_i} L_i(\theta_i)$$
$$\nabla_{\theta_i} L_i(\theta_i) = -\,\mathbb{E}_{(s_t,a_t,r_t,s_{t+1},d_t)\sim U(D)}[(y_i - Q(s_t, a_t; \theta_i))\nabla_{\theta_i} Q(s_t, a_t; \theta_i)],$$

with a learning α which was dynamically adapted by the optimized Adam gradient descent of `Tensorflow Keras`.

Since this is a relatively low-dimensional problem, a multi-layer perceptron with two fully connected 256-neuron layers, each followed by linear rectification and a final 4-dimensional linear output was a sufficiently powerful architecture. This rather wide than deep architecture was inspired by Yu (2019). Although the Q-function approximates Q-values as a function of state-action pairs, the actions were not given as input, but their Q-values are represented by the corresponding output neuron value.

3.2.1 Exploration

The learning algorithm uses a greedy strategy $a = \max_a Q(s, a; \theta)$ during weight updates, thus, an epsilon-greedy policy for action selection must ensure that the state space is properly explored. Two ϵ decays were compared.

Linear Condition Firstly, ϵ was linearly decreased at each experience replay episode by 0.002 from 1 to 0.01, continuously decreasing over the first 495 episodes, resulting in exploratory random actions at the start of learning, then gradual convergence to an optimum.

Exponential Condition Secondly, ϵ was exponentially decayed at each experience replay episode by a factor of 0.99 from 1 to 0.01, resulting in a smoother but steeper transition from exploration to exploitation.

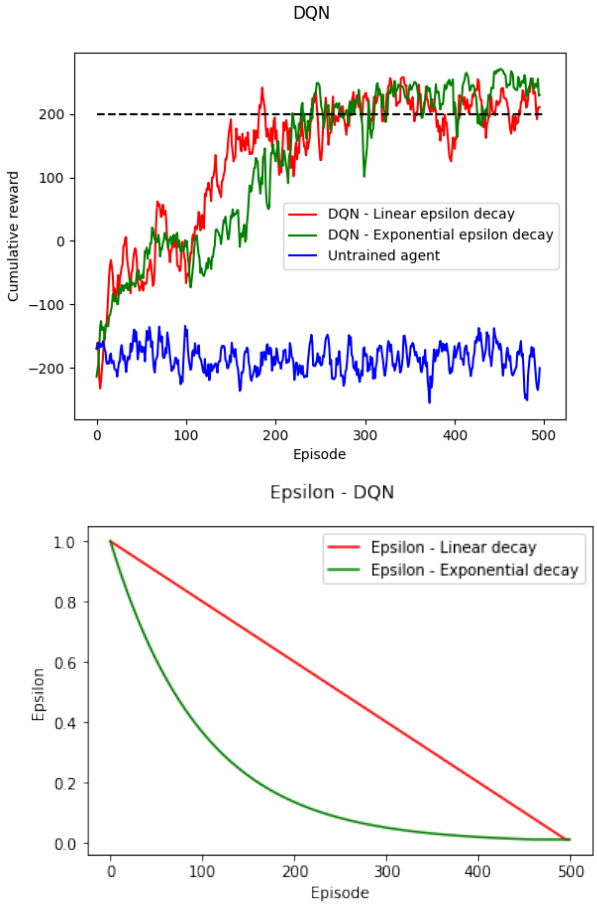

Figure 3: The top figure shows cumulative reward per episode for 500 training episodes, averaged over 5 experiments for DQN with linear epsilon decay (red) and DQN with exponential epsilon decay (green). In addition, a random agent's performance (blue) is added for reference and the goal of 200 cumulative reward per episode is marked with a dashed line. The respective values of epsilon on both conditions are plotted on the bottom figure.

3.2.2 Experiment

For both conditions, a total of 5 repetitions were performed with 500 training episodes each. The scores (cumulative reward per episode) were average over the runs. The following hyper-parameters were used: $\epsilon = 1$ (amount of exploration), $\gamma = .99$ (temporal discount), $\alpha = 0.001$ (learning rate), $\tau = 0.01$ (update rate of target network), $|D| = 500000$ (memory buffer size, number of experiences possibly stored at once), $|B| = 64$ (batch size). These orders of magnitude of these hyper-parameters were inspired by those used in previous studies described in Section 2.

3.2.3 Results of Deep Q-Learning

The results of the experiment are shown in Figure 3. The mean scores converged at around 300 episodes, and there was no visible difference between the two conditions. Both DQN algorithms hit the mark of > 200 cumulative reward.

4 Discussion

This study aimed (a) to solve the Lunar Lander problem with the Sarsa and Deep Q-learning algorithms and (b) to explore the effects of different ϵ-greedy schedules on the performance of these algorithms.

A successfully trained agent was defined as one that converged to a policy that, when followed, results in coming to rest at the landing pad (around (0,0)) most of the time. This translates to a mean score or cumulative reward of above 200.

Firstly, although Sarsa agents with discretization strategies 2.1 and 2.2 performed considerably better than a random agent, they were not successful. Post-hoc tests did not show any clear improvements after the 10000 epochs of the experiment. This is most likely due to the very large complexity and, consequently, the low ratio between Q-values and samples made any training until convergence unfeasible.

Furthermore, it was found that adopting a more refined discretization strategy and optimizing the exploration-defining parameter ϵ as a function of time leads to performance improvements. The exploratory effects of a large ϵ can also be observed in the added variance of the scores (Figure 2). This shows the importance of adapting hyper-parameters to the environments at hand.

Moreover, resulting performance of the Sarsa agent was worse than in the findings from Gadgil et al. (2020); their more refined discretization method led to successful convergence to a score of 150. This suggests that discretizing the state space with an even more refined strategy may further improve the performance of the Sarsa algorithm.

The DQN agents from the linear and exponential condition outperformed the discretized approach and the random agent and reach the mark for success, converging to a score of above 200 around 300 episodes. The lack of a visible difference in performance between the conditions suggests that the added exploration in the linear ϵ-schedule may not have a significant effect on the performance of this DQN.

An idea for future improvement would be to apply the DSQN algorithm, as described by Xu et al. (2018), to the Lunar Lander Problem. This method would add exploration during weight updates instead of during the agent's interaction with the environment.

References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Gadgil, S., Xin, Y., & Xu, C. (2020). Solving the lunar lander problem under uncertainty using reinforcement learning. In *2020 southeastcon* (Vol. 2, pp. 1–8).

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Xu, Z.-x., Cao, L., Chen, X.-l., Li, C.-x., Zhang, Y.-l., & Lai, J. (2018). Deep reinforcement learning with sarsa and q-learning: A hybrid approach. *IEICE TRANSACTIONS on Information and Systems*, *101*(9), 2315–2322.

Yu, X. (2019). Deep q-learning on lunar lander game. *Georgia Institute of Technology*.