Short Reinforcement Learning Project: Lunar Lander

Cody Black (5033) and Reza Niazi (5033)

Abstract

In this paper we will discuss applications of reinforcement learning to Open AI Gym's Lunar Lander interface. We discuss a series of hypotheses, experiments to test the hypotheses, and results. Our main goal was to compare agents that use RL techniques and see if they could outperform a heuristic agent. We do not expect this to be possible, as we will be applying coarse coding techniques to 6 continuous state space parameters. Nevertheless, this simplified model is able to achieve a somewhat significant performance compared to the heuristic.

Project Domain

The default Lunar Lander Open AI Gym environment emulates the landing of a spacecraft on the surface of the moon. The spacecraft has an action space of

$\mathcal{A} = \{\text{Do nothing}, \text{Fire left orientation engine}, \text{Fire main engine}, \text{Fire right orientation engine}\}.$

The state space is continuous, and has 8 parameters: horizontal/vertical position, horizontal/vertical velocity, angle, angular velocity, and left/right leg contact (Boolean). Among these, 6 are continuous. An episode ends when the lander crashes or lands on the surface. Landing anywhere on the surface is allowed, but ideally it will land in a designated landing zone, with coordinates (0,0). For every episode, the lander is initialized at the top of the screen and with random values for the other parameters.

Literature Review

Given that we have a continuous state space, we will use discretization/coarse coding to create partitions of the different parameters in the state space. This can be viewed as a special case of Tile Coding[1],[2]. Though [1] and [2] look at one and multiple layers in their tile coding approach, we will just look at one layer. Convergence to a desired performance is achieved in all cases, but of course it is quicker the more layers are added.

We would like to set a benchmark on just how well one layer can do, especially because our problem has many more continuous parameters than in [1] and [2]. For example, the gridworld and mountain car problems in [1] and [2], respectively, only have 2 continuous parameters, but in our more complex example there are 6.

Since [1] and [2] do not address the lunar lander, we will refer to [3] and [4] for this. These papers use the more sophisticated neural network approach to train the lunar lander. Therefore, we can see just how well the simplified 1 layer of tiling does against the more sophisticated function approximation approach. In fact, to our knowledge no one has yet to explore how well the coarse coding process works for lunar lander, as most have dealt with the more simplified problems as in [1] and [2].

Experiment Review

Hypotheses:
The lander comes with a built in heuristic, and therefore our first and second hypotheses were that Q-Learning and SARSA could become at least half as effective as the heuristic on the default Lunar Lander environment. We chose Q-Learning and SARSA, as these are both model free methods. However, Q-Learning is an off policy algorithm, whereas SARSA is on policy. Therefore, we wanted to see how these would compare and contrast.

We did not expect an agent using reinforcement learning to beat the heuristic in the default environment, as we were discretizing 6 continuous parameters with just one layer of tiling. However, for our third and fourth hypotheses we used a modified Lunar Lander environment where the landing zone moved from left to right over a flat landscape. The heuristic performs poorly in this environment, so we hypothesized that Q-learning and SARSA could perform better than the heuristic in this environment.

Implementation:
Each reinforcement learning method was used to train an agent for a certain number of episodes of each environment. Although the performance of the RL agents continued to improve over time, eventually the rate of improvement stagnated. We found that after 5,000 episodes, the performance of the RL agents did not increase significantly, and stopped our learning at this point. Theoretically, we could train the lander for more episodes, but this would be computationally infeasible for us.

In order to use Q-learning and SARSA on the continuous state space, we converted each continuous state into a corresponding discrete state. To do so, we sorted each component of the state vector into discrete bins. This is essentially making a certain range of state values to be equivalent in each of the 6 continuous dimensions. The combined indices of each discrete bin were used to find the index of the q-table that corresponded to the continuous state. Q-Learning was handled by Cody and SARSA by Reza. For the discretization process, each of the 6 continuous parameters were partitioned into 11 bins. Also, in all experiments, the learning rate $\alpha = 0.1$, discount factor $\gamma = 0.6$, and rate of taking non-greedy action $\epsilon = .1$

Evaluation Metrics:
The Q-learning, SARSA, and heuristic agents were evaluated using their average total reward and their success rate. For each episode, the total reward is the sum of the reward given at each timestep. The reward function used is based on the default Lunar Lander reward function, which increases reward when the lander moves closer to the landing zone and decreases reward proportional to the magnitude of the lander's velocity and angle. The reward is also slightly decreased every time the agent fires an engine. This means that a more efficient agent has a higher total reward. The reward

function was modified to add a small penalty proportional to the magnitude of the lander's angular velocity.

Additionally, each episode may be marked as successful. A successful episode is one where the lander successfully lands anywhere on the surface without crashing. Although the lander does not have to land in the landing zone for the episode to be counted as successful, landing outside of the landing zone results in a lower total reward value overall.

Results:
Experiments 1 and 2 involved using Q-learning and SARSA to train an agent on the default Lunar Lander environment. Both agents had the most improvement in the first 1,000 episodes, and after 5,000 episodes, the average reward and success rate were relatively stable. Both SARSA and Q-learning ended with an average reward value of around 100, and the average success rate for both SARSA and Q-learning was around 60%. These results are shown in Figure 1 below.

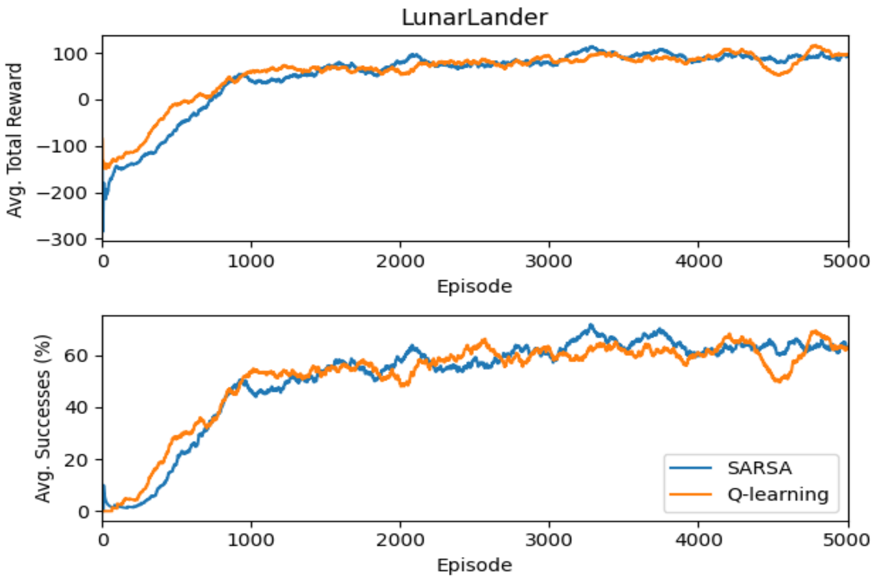

Figure 1: Average reward and success rate for an agent using SARSA or Q-learning over 5,000 episodes of the default Lunar Lander environment. For each episode, the average is calculated as the average of the last 250 episodes.

Experiments 3 and 4 used the modified Lunar Lander environment, where the landing zone moved from left to right over a flat surface. Like the first two experiments, the average reward improved the most in the first 1,000 episodes and was relatively stable by episode 5,000. SARSA ended with a reward of around 250, and Q-learning ended with a reward of around 200. However, unlike the first two experiments, the average success rate seemed to vary substantially. Both SARSA and Q-learning reached a maximum success rate of around 30%, but SARSA's success rate dropped to around 20% and Q-learning's success rate dropped to around 10% by episode 5,000. These results are shown below in Figure 2.

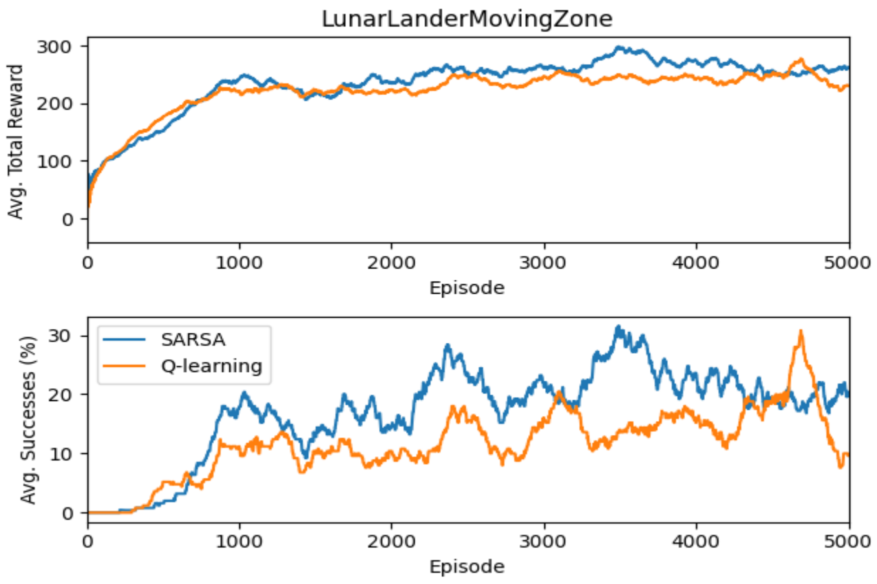

Figure 2: Average reward and success rate for an agent using SARSA or Q-learning over 5,000 episodes of the Lunar Lander environment with the moving landing zone. For each episode, the average is calculated as the average of the last 250 episodes.

The RL agents were also evaluated by using the q-table generated by each learning method to run 500 additional episodes where no more learning took place. This was done in order to evaluate the policy generated by each learning method. The heuristic also completed 500 episodes in each environment, and its average total reward and success rate were recorded. These results are summarized in Table 1 below.

Lunar Lander environment	Method	Evaluation after learning		5000th episode of learning	
		Avg. Reward	Success Rate	Avg. Reward	Success Rate
Default	Heuristic	237.85	92.2%	N/A	N/A
	SARSA	-45.58	35.0%	≈100	≈60%
	Q-learning	13.71	50.2%	≈100	≈60%
Moving landing zone	Heuristic	304.18	36.6%	N/A	N/A
	SARSA	208.72	18.8%	≈250	≈20%
	Q-learning	152.12	0.2%	≈200	≈10%

Table 1: Summary of results for the Heuristic and each RL method in each environment.

Analysis:
Based on reward alone, none of the RL agents performed as well as we hypothesised they would. In the first two experiments, the average total reward was less than half that of the heuristic in all

cases, and in the second two experiments, the reward of the RL agents did not surpass that of the heuristic, although SARSA was somewhat close to doing so. However, the success rate suggests that the agents in the first two experiments likely performed better than the average total reward suggests. This indicates that poor performance of the RL agents may be at least partially caused by the way the reward function is calculated.

In all four experiments, the agents seemed to perform worse when acting only based off of the learned policy. This is likely because our discretization of the state space resulted in a very large number of discrete states. After 5,000 episodes, less than 1% of the discrete states had an associated Q value that was not zero (i.e. a Q value that had changed from the initial value of 0). This means that an agent using a policy derived from the q-table would likely experience states where it simply had to choose the default action because it did not have any relevant information in the q-table to use to choose a better action for that state. This is also another likely reason why the performance of the RL agents was worse than expected.

When the landing zone was in a fixed location, such as in the default Lunar Lander environment, SARSA and Q-learning had very similar learning curves. When the landing zone was moving, Q-learning's success rate was almost constantly less than SARSA's success rate, except for a large spike near the end. One possible explanation for this is that SARSA is more likely to choose "safer" actions that are less optimal but are more likely to avoid the large reward penalty for crashing. Q-learning may be choosing more optimal actions, such as flying towards the landing zone to quickly increase its total reward, but crashing more often as a result.

Comparison with Previous Work

We will first compare how our coarse coding with 6 continuous parameters compares to the 2 continuous parameter discretization done in [1] and [2]. In [1], Q-Learning was only used, and they were applying it to various amounts of tiling. They wanted to see how varying a parameter would change their results. In the worst case, they were able to achieve a reward that was 40% the optimal one. For the regular environment, we can see that both Q-Learning and SARSA were able to give us around 50% the optimal performance. In [2], the performance is measured by how many steps it will take a mountain car to reach the top of a hill from a trough. Given the optimal choice of algorithm and parameters, they were able to get an average of 450 steps to complete the task. In the worst case scenario, this was closer to 700. In our example, a rough estimate of the number of steps per episode for our lunar lander was 692. This is not too far of the approach in [2]

Now, we can compare how Deep Q-Learning compared to Q-Learning and SARSA. In [3], regardless of parameter tuning, the agent was able to achieve a 100-episode-mean of at least 200 by 2000 episodes. Of course, this outperforms our agent,but this is to be expected. In [4], they did not experiment with parameters and only showed their best result. In this case, they were able to achieve a mean reward of at least 200 by 700 episodes.

Conclusion

Given that we had an 8 dimensional state space with 6 continuous values, the process of coarse coding produced non-negligible results through Q-Learning and SARSA. Though the RL agents did not perform as well as we expected relative to the heuristic in either of the environments, they were able to learn how to land at an appreciable rate. The results could be improved with more complexity in the discretization process and with more computational power. Future work can be done on improving the discretization process, and applying more sophisticated versions of our current algorithms like SARSA(λ).

References

[1] Alexander A. Sherstov and Peter Stone. *Function Approximation via Tile Coding: Automating Parameter Choice.* In Proc. Symposium on Abstraction, Reformulation, and Approximation (SARA-05)

[2] Richard S. Sutton *Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding* . Advances in Neural Information Processing Systems 8 (Pages 1038-1044)

[3] Xinli Yu. *Deep Q-Learning on Lunar Lander Game.* Technical Report

[4] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat *Deep Q-Network Based Multi-agent Reinforcement Learning with Binary Action Agents* arXiv:2008.04109