COMP ENG 4SL4: Machine learning

Assignment 5 - Reinforcement Learning Hackathon 2024

Instructor: Dr. Sorina Dumitrescu

TA: Hazem Taha

Richard Qiu – qiur12 – 400318681

Abdul Nasir Noori – nooria10 – 400325387

12/01/2024

Introduction

The objective of this assignment is to develop and train a reinforcement learning agent, particularly a lunar lander, tasked to navigate and land safely between the goal posts with minimal human guidance. Any reinforcement learning algorithm can be explored to implement the model; the goal is to train the best model and achieve the highest reward possible. We investigate two methods where the first one (Tabular Q-learning) works well in the discrete space by utilizing the Q-tables updates and the second method (D-QN) by implementing a neural network and memory buffer to strength the Q-learning in continuous space. We also investigate the impact of hyperparameters, modifications to the base algorithms, and training techniques to enhance learning efficiency and stability.

Methodology

The reinforcement learning algorithm chosen for the implementation was the Q-learning algorithm. The flow of the algorithm is as follows; the Q-table is initialized with zeros, an epsilon-greedy strategy is used for the agent to take an action to either explore (chose a random action) or exploit based on the exploration rate (ε), and the Q-table is updated correspondingly. In training the model, at each episode (epoch), an epsilon decay is utilized to reduce the exploration rate until it reaches a set minimum. Once the minimum exploration rate has been reached, the algorithm checks if the current average performance of the model is better than the best average performance, if so, it updates it correspondingly and saves the model as the best model.

To further explore the Q-learning method in continuous space environments. We also experiment implementing the DQN (Deep Q-Networks) to handle a more complex environment by following the Pytorch tutorial. In DQN, a neural network is used to get the state from the input layer and outputs the actions in the last layer. A replay memory buffer is used to store the previous state values, and for each state, a random batch size (64) of state values will be picked from the memory buffer to participate the Q-learning updates, for the training update rule. The Q function obeys the Bellman equation as shown below:

$$Q^{\pi}(s,a) = r + \gamma Q^{\pi}(s', \pi(s'))$$

And the temporal difference error between the two Q values is calculated as follows:

$$\delta = Q(s,a) - (r + \gamma \max_{a'} Q(s',a))$$

To minimize the training error, we research and decide to use the Huber loss where it acts like the mean squared error when the error is small, but like the mean absolute error when the error is large which it can help quickly converge when the reward of Q is bad[1]. The loss formula is shown below.

$$\mathcal{L} = \frac{1}{|B|} \sum_{(s,a,s',r) \in B} \mathcal{L}(\delta)$$

$$\text{where} \quad \mathcal{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases}$$

The training process contains 1000 iterations over the lander environment and an epsilon-greedy policy for action selection is implemented where the exploration rate epsilon would decay over time.

```
self.epsilon = max(self.epsilon_min,self.epsilon * self.epsilon_decay)
```

The best collected testing reward is 273 with the DQN method with the following parameters.

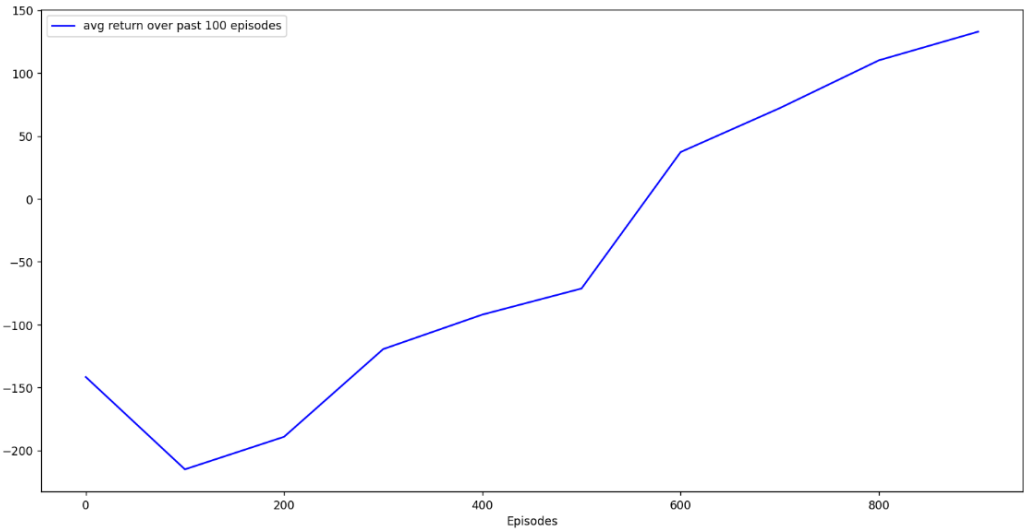
Alpha	5e-4
Gamma	0.99
Epsilon_start	1.0
Epsilon_decay_rate	0.995
Epsilon_min	0.01
Tau	1e-3
Batch_size	64
Network_layer	3
Layer_size	128

Moreover, after several research, a soft update method is used to gradually update the weights of a target network by blending its parameters with the parameters of the policy network (or online network). This method provides smoother and more stable updates to the target network compare to the hard updates which copy the policy network to the target network directly [2].

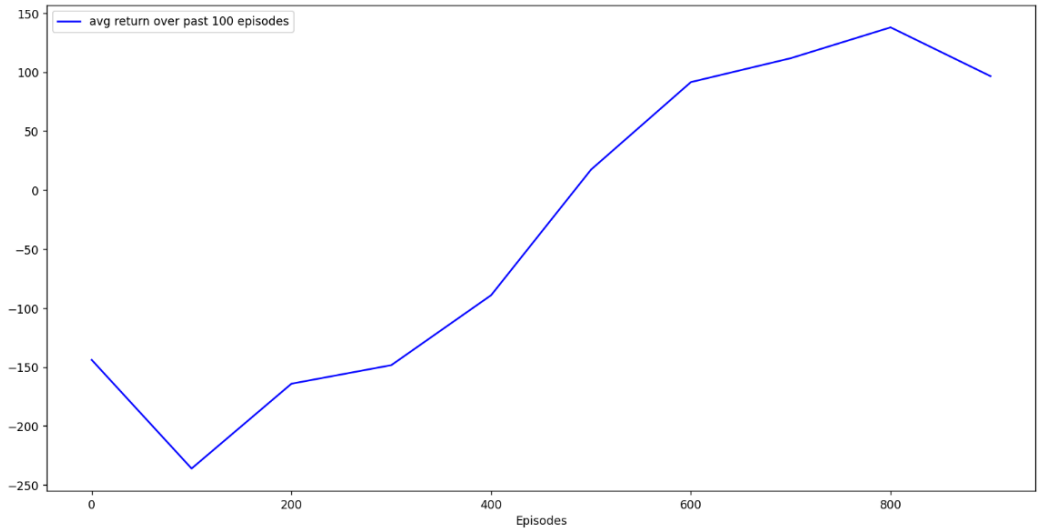
Results

Due to the time constraints, we only collected the Tabular Q-learning graphs but the DQN follows the similar trend.

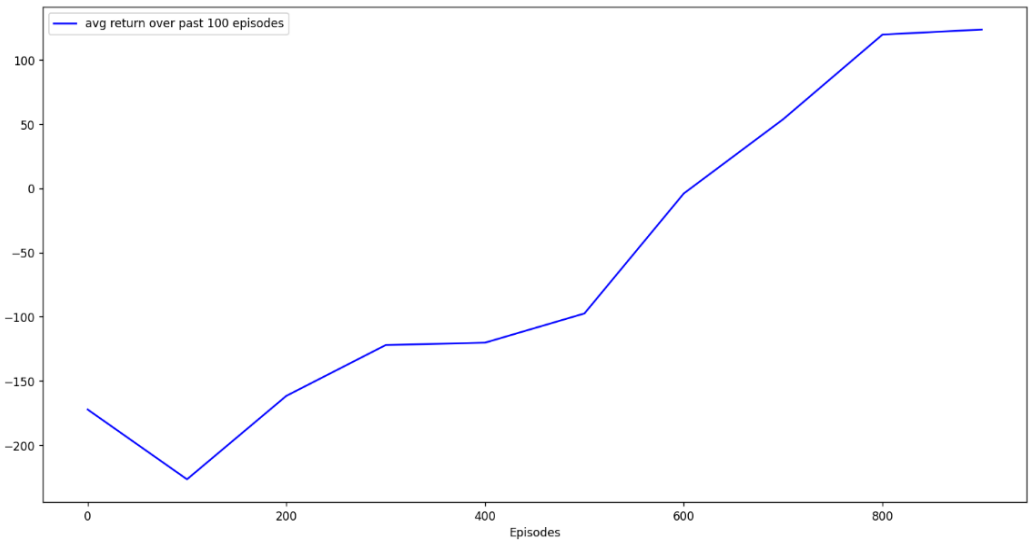
1. Configuration: Exploration Rate (ε) = 1.0



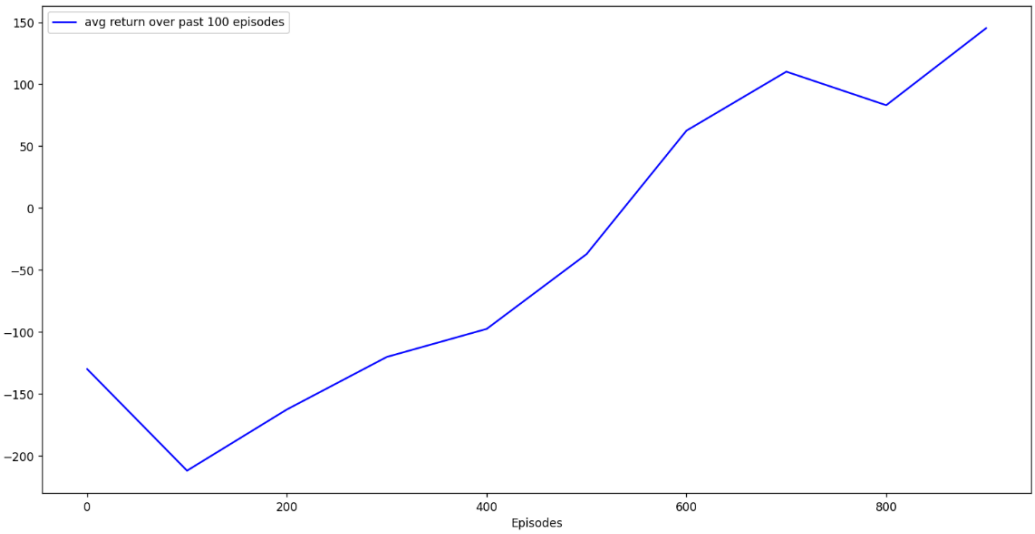
2. Configuration: Exploration Rate (ε) = 0.9



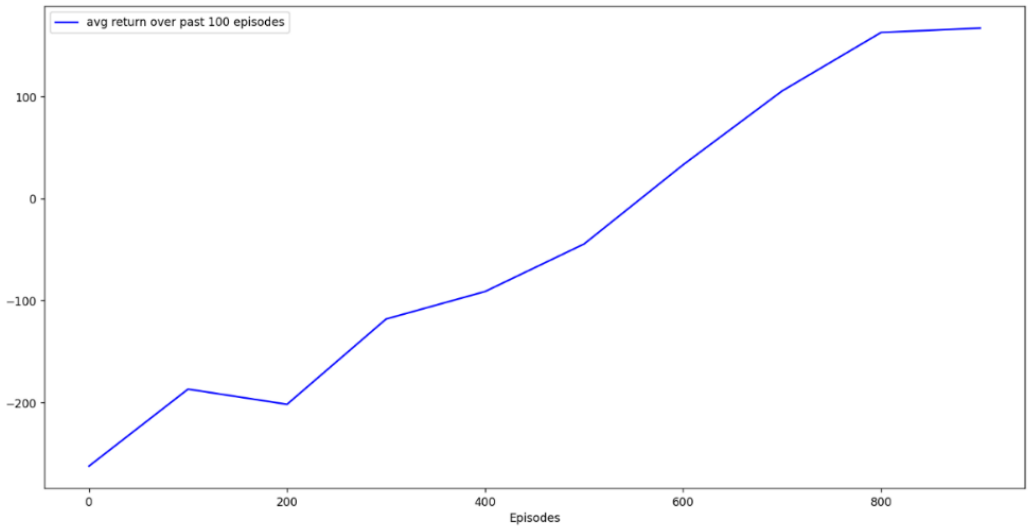
3. Configuration: Exploration Rate (ε) = 0.7



4. Configuration: Exploration Rate (ε) = 0.5



5. Configuration: Exploration Rate (ε) = 0.1



Discussion

To generate the results, the exploration rate (ε) was varied from high values to lower values. The idea behind the exploration rate is that having a higher value means it focuses more on learning new information to discover a good policy, while a lower value means it focuses more on exploitation with the known knowledge to maximize rewards. From the results, we can observe that as we lower the "starting" exploration rate (ε), the average return over past 100 episodes doesn't seem to change in a consistent way, rather we can see that sometimes it performs almost the same or slightly better, like with ε = 0.9 and ε = 0.5, while other times it performs poorly, like with ε = 0.7 and ε = 0.1. This behavior is expected, since having lower exploration (ε) focuses more quickly on exploiting, thus it exhibits a more random behavior, either it may find a good policy (good performance) or a not so good one (poor performance).

Summary of Hyperparameter Impacts.

Hyperparameter	Tabular Q-Learning	DQN
Learning Rate	Controls the speed of Q-table; too high can destabilize	Affects Weight Updates; too high can lead to divergence
Discount Factor	Balance short term vs long-term rewards	High values improve long-term planning.

Exploration Rate	Trade off between convergence	Small values will ensure gradual transition to exploitation
Batch Size	NA	Large batches improve stability but will slow the performance
Replay Memory Size	NA	Larger batches enhance diversity and stability
Target Network Updates	NA	Frequent updates can destabilize; soft updates work well.
Network layer	NA	Larger networks improve function approximation.

Conclusion

In conclusion, we explored the application of Q-learning techniques in both tabular and Deep Q-Networks (DQN), to train an agent to land successfully in the Lunar Lander environment. By examining the effects of different exploration rates (ε) in tabular Q-learning, we observed the trade-off between exploration and exploitation, highlighting the impact on the agent's performance.

DQN proved to be a powerful method for tackling continuous state spaces by leveraging a neural network and replay memory buffer for effective learning. We also experienced and found out the importance of hyperparameter tuning and training techniques in reinforcement learning, such as soft updates for the target network and find the best epsilon and learning rate.

Future work could be more tunning and make the network better to enhance the agent’s performance. Overall we gain a deep understanding in both Q-learning and DQN within this project.

Reference

[1] “Reinforcement learning (DQN) tutorial¶,” Reinforcement Learning (DQN) Tutorial - PyTorch Tutorials 2.5.0+cu124 documentation, https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html#training (accessed Dec. 1, 2024).

[2] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” arXiv.org, https://arxiv.org/abs/1509.02971 (accessed Dec. 1, 2024).