Reinforcement Learning EL2805 - Lab 2

Rodrigo Montero (20040421-T054)
Daniel De Dios Allegue (20041029-T074)

December 16, 2024

Part 1 - Deep Q-Network (DQN)

Introduction

In this section of the report, we detail our implementation of a Deep Q-Network (DQN) to solve the Lunar Lander environment. The problem involves training an agent to land a lunar module on a designated pad, maximizing cumulative rewards. The solution is evaluated based on the agent's ability to achieve an average total reward of at least 50 points over the last 50 episodes.

Key Components in DQN

Replay Buffer

The replay buffer stores the agent's experiences (state, action, reward, next state, and done). By sampling random batches for training, it breaks the temporal correlation in the agent's experiences and improves stability in updates. Our buffer can hold up to 10,000 experiences, with a batch size of 32 for training.

Target Network

The target network stabilizes training by providing consistent Q-value targets over multiple updates. It is updated periodically (every 10 episodes in our implementation) by copying weights from the main Q-network.

Implementation Details

Experience Replay Buffer

We implemented a custom replay buffer to store experiences. It allows random sampling and dynamically maintains a fixed maximum size of 10,000 experiences.

Neural Network Architecture

- **Input Layer**: Accepts the environment's state vector (dimension: 8).
- **Hidden Layers**: Two fully connected layers with 32 units each, activated using ReLU.
- **Output Layer**: Outputs Q-values for all actions (dimension: 4).

This architecture balances simplicity and effectiveness for the Lunar Lander problem.

Optimizer

We used the Adam optimizer with a learning rate of 5×10^{-4}. Adam was chosen for its adaptive learning rate capabilities, which are suitable for non-stationary problems like reinforcement learning.

Parameter Choices

- $\gamma = 0.9$: The discount factor prioritizes short-term rewards while maintaining sensitivity to future outcomes.
- $\epsilon = 0.1$: A small exploration rate balances exploration and exploitation.
- **Batch Size**: 16, chosen for efficient gradient updates without excessive memory usage.
- **Replay Buffer Size**: 10,000, providing sufficient diversity in experiences.
- **Target Update Frequency** (C): Every 200 episodes.

Performance and Modifications

To meet the performance benchmark, we implemented the standard DQN algorithm without additional modifications like Double DQN, Dueling DQN, or Combined Experience Replay. The agent successfully achieved an average reward of over 50.

Analysis and Results

Training Process

To analyze the training process, we plotted the total episodic reward and the total number of steps taken per episode during training (see Figure 1). While the individual rewards fluctuated significantly, the running average showed a steady increase until it reached an average reward of 100. At this point, we applied an early stopping criterion, considering this reward satisfactory. This suggests that the agent successfully learned to solve the task over the course of training.

Effect of Discount Factor

We investigated the effect of the discount factor γ on training by testing three values: $\gamma_0 = 0.9$, $\gamma_1 = 1.0$, and $\gamma_2 = 0.5$.

Both $\gamma_0 = 0.9$ and $\gamma_1 = 1.0$ produced similar training performance, with positive rewards observed after training. This suggests that these values allow the agent to effectively balance immediate and long-term rewards. However, $\gamma_2 = 0.5$ resulted in predominantly negative rewards, indicating that a smaller discount factor reduces the agent's ability to consider long-term rewards effectively. The results are shown in Figures 2, 3, and 4.

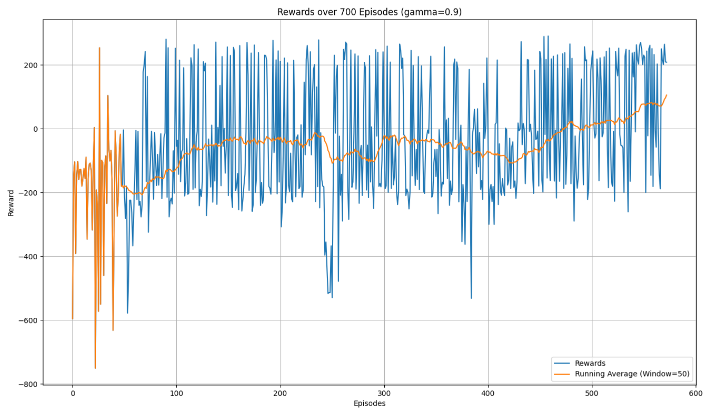


Figure 1: Total episodic reward and total steps per episode during training.

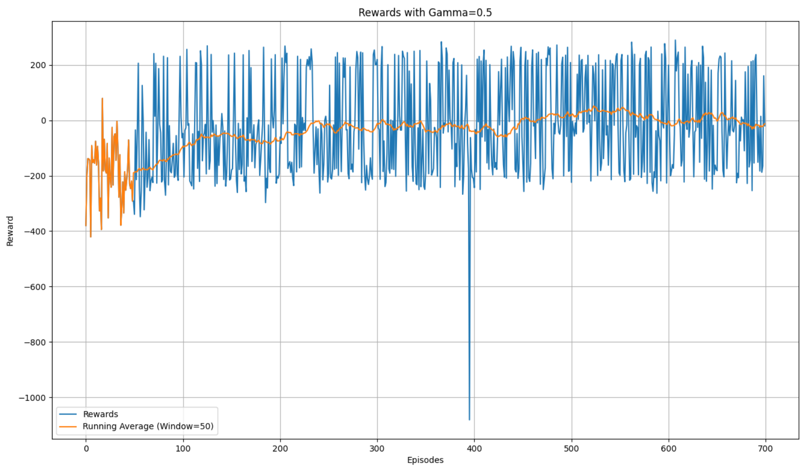


Figure 2: Episodic rewards over time for $\gamma = 0.5$.

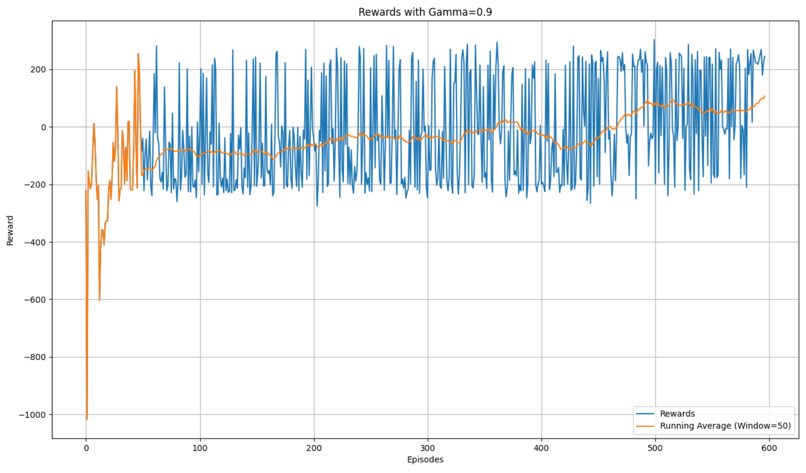


Figure 3: Episodic rewards over time for $\gamma = 0.9$.

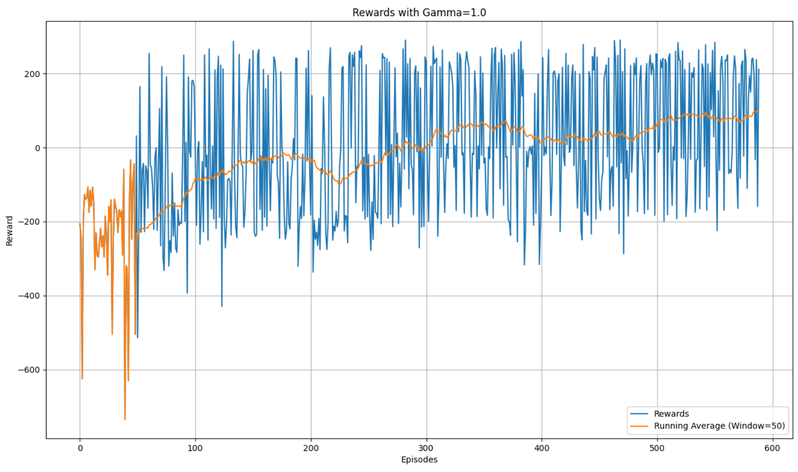


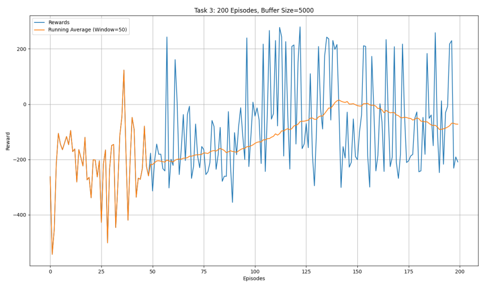
Figure 4: Episodic rewards over time for $\gamma = 1.0$.

Effect of Episodes and Buffer Size

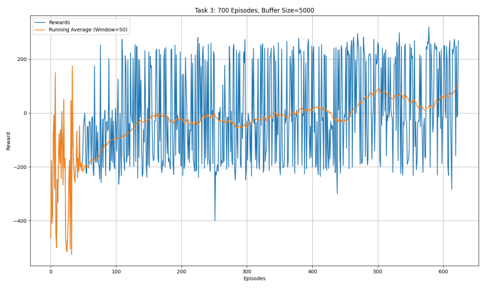
We explored the impact of varying the number of training episodes and the replay buffer size on training performance. To prevent clutter and maintain clarity, we selected four representative plots that illustrate the main conclusions (Figure 5). These plots focus on the key observations regarding the number of episodes and buffer sizes.

From these results, we observed the following:

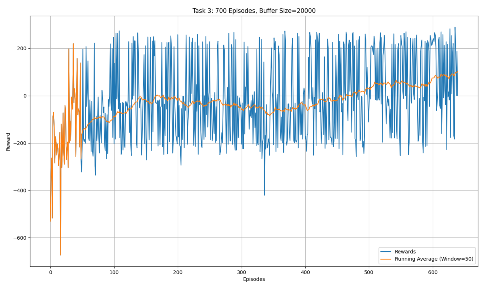
- **Number of Episodes:** Training with 200 episodes was insufficient for the agent to reach the target average reward of 100, highlighting the importance of longer training durations. In contrast, both 700 and 1000 episodes allowed the model to reach the target reward, with early stopping once the threshold was met.
- **Buffer Size:** No significant differences were noted between buffer sizes of 5000, 10000, and 20000. This indicates that, within this range, buffer size did not substantially affect the model's learning efficiency or performance.



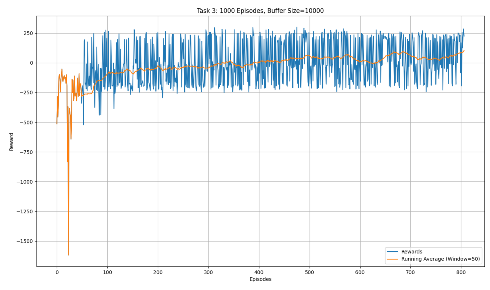
(a) 200 episodes, buffer size = 5000



(b) 700 episodes, buffer size = 5000



(c) 700 episodes, buffer size = 20000



(d) 1000 episodes, buffer size = 10000

Figure 5: Representative plots showing the effect of varying the number of episodes and replay buffer sizes on training performance.

Optimal Policy Evaluation

Max Q-values

Figure 6 shows the maximum Q-values ($\max_a Q_\theta(s(y, \omega), a)$) for the restricted state $s(y, \omega) = (0, y, 0, 0, \omega, 0, 0, 0)$, where y represents the height of the lander, and ω represents its angle. The plot reveals that the Q-values are highest when ω is close to 0, indicating the policy prioritizes maintaining the lander in an upright position. Additionally, lower values of y (indicating proximity to the ground) are associated with higher Q-values, suggesting the policy considers stability near landing critical.

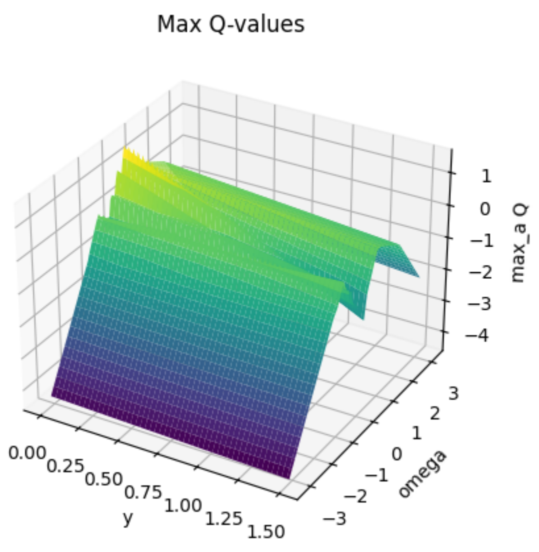


Figure 6: Maximum Q-values for varying y and ω.

Optimal Actions

The optimal actions $\arg\max_a Q_\theta(s(y, \omega), a)$ for varying y and ω are presented in Figure 7. The 2D color plot indicates the following behavior:

- For positive ω, the best action is firing the right engine (Action 3).
- For negative ω, the best action is firing the left engine (Action 1).
- When ω is close to 0 (and slightly negative), the main engine (Action 2) is often selected.
- Action 0 (do nothing) is rarely chosen.

This behavior aligns with the objective of stabilizing the lander's angle and descent.

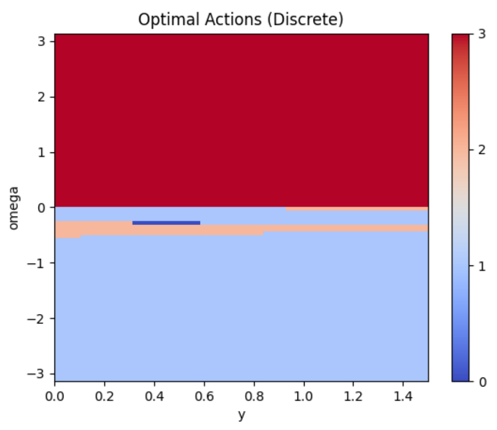


Figure 7: Optimal actions for varying y and ω.

Comparison with Random Agent

To compare the Q-network with a random agent, we plotted the total episodic reward over 50 episodes for both agents (Figure 8). The random agent consistently achieved a reward near -200, reflecting poor performance. In contrast, the trained agent reached positive rewards in most episodes, demonstrating its effectiveness in solving the task. Occasionally, the trained agent achieved abnormally low rewards, possibly due to exploration or outlier scenarios.

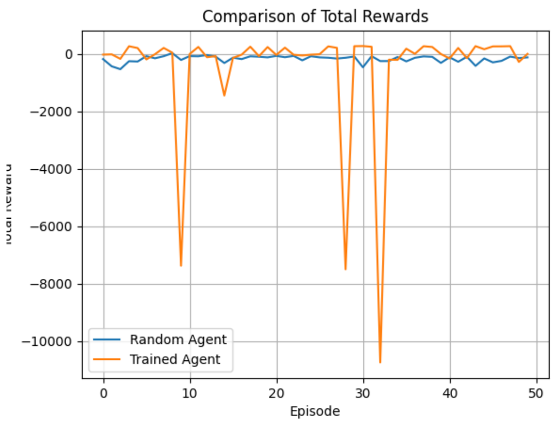


Figure 8: Comparison of episodic rewards for the Q-network and random agent.

Part 2 - Deep Deterministic Policy Gradient (DDPG)

Task Description

The aim of this section was to implement and analyze the Deep Deterministic Policy Gradient (DDPG) algorithm to solve the LunarLanderContinuous-v3 environment by implementing the neural network architecture, training the agent, and performing experiments to evaluate the effects of hyperparameters and other factors on training performance.

Implementation Details

Baseline Agent

To establish a baseline, we used a random agent that selects actions uniformly in the range $[-1, 1]$. This agent serves as a benchmark for comparing the performance of the trained DDPG agent.

Network Architecture and Training Setup

We implemented the actor and critic networks with the following specifications:

- **Actor Network:** Three fully connected layers. The first layer has 400 neurons with ReLU activation, the second layer has 200 neurons with ReLU activation, and the output layer has two neurons with a `tanh` activation to constrain actions to $[-1, 1]$.

- **Critic Network:** Similar to the actor network, but with the state as input to the first layer. The state and action vectors are concatenated at the output of the first layer before being passed to subsequent layers.

We used the Adam optimizer for both networks with learning rates $5 \cdot 10^{-5}$ for the actor and $5 \cdot 10^{-4}$ for the critic. Gradient norm clipping with a 2-norm limit of 1 was applied during backpropagation.

Other training parameters included:

- Discount factor: $\gamma = 0.99$
- Replay buffer size: $L = 30,000$
- Batch size: $N = 64$
- Noise parameters: $\mu = 0.15$, $\sigma = 0.2$
- Target network update parameter: $\tau = 10^{-3}$

Key Design Choices

The actor network's lower learning rate reflects the need for a stable policy update process, as the actor directly influences exploration. Conversely, the critic has a higher learning rate since it estimates value functions, requiring faster adaptation to policy changes. The target networks stabilize training by slowly tracking the main networks using soft updates.

Critic vs. Actor Learning Rate

It is generally better for the critic to have a larger learning rate than the actor. The critic must adapt quickly to policy changes to provide accurate value estimates, while the actor requires a smaller learning rate to ensure stable and gradual policy updates. This balance prevents instability and allows the actor and critic networks to complement each other effectively during training.

Experimental Results

Training Performance

Figure 9 shows the total episodic reward and steps per episode over 400 training episodes. The agent quickly learns to achieve positive rewards, with rapid improvement early in training. Progress slows after reaching positive rewards, with gradual increases over time. Episodes with lower rewards tend to have fewer steps, indicating quicker terminations due to suboptimal policies. Overall, the agent demonstrates successful learning, achieving consistent positive rewards.

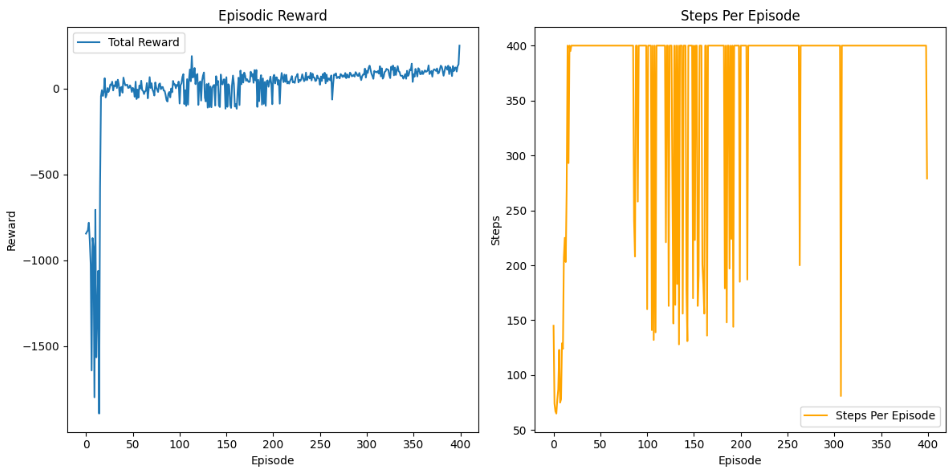


Figure 9: Episodic reward and steps per episode during training.

Effect of Discount Factor

We evaluated the effect of different discount factors, $\gamma_1 = 1.0$, $\gamma_0 = 0.99$, and $\gamma_2 = 0.5$, while keeping other parameters constant. The results in Figure 10 show the following:

- Both $\gamma_1 = 1.0$ and $\gamma_0 = 0.99$ showed similar results, with stable and positive performance, and the agent consistently achieved near the required average reward threshold.
- $\gamma_2 = 0.5$ initially performed similarly to the other values but eventually started to decline, with higher variance and worse results as training progressed. This suggests that a lower discount factor leads to more worse decision-making and instability.

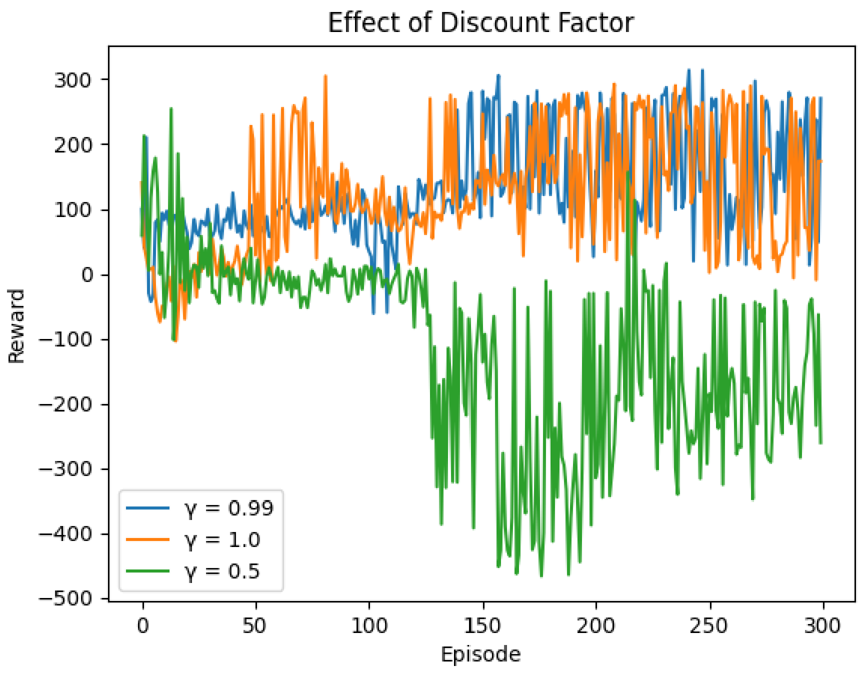


Figure 10: Effect of discount factor on training performance.

Effect of Replay Buffer Size

We examined the impact of different replay buffer sizes, specifically $L = 15,000$ and $L = 50,000$ and the resultscan be seen in Figure 11. Both buffer sizes yielded comparable performance, with rewards gradually improving over time. Overall, the difference in performance between the two buffer sizes was minimal, and neither configuration displayed a significant advantage over the other in terms of training stability or reward accumulation.

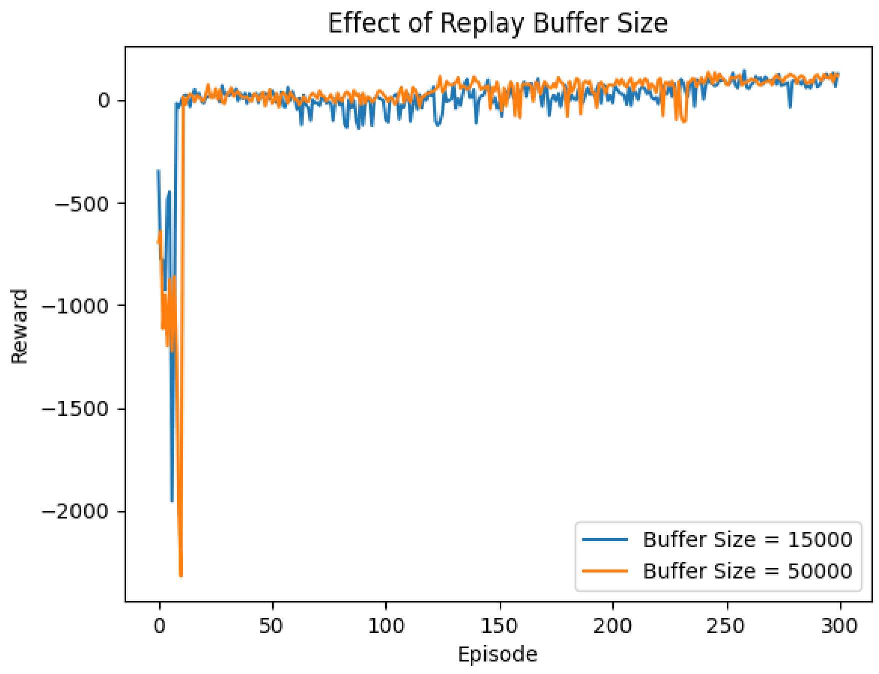


Figure 11: Effect of replay buffer size on training performance.

Q-value and Policy Visualization

Figure 12 shows the Q-value surface for the restricted state $s(y, \omega)$. The Q-value appears to be more sensitive to changes in the angle ω, with values being lower near $\omega = 0$ and increasing as the angle deviates from zero, both positively and negatively. Height y has a less pronounced effect on the Q-value, although a slight increase in height might lead to a marginally higher Q-value.

Figure 13 plots the engine direction $\pi_\theta(s(y, \omega))_2$. The engine direction is clearly influenced by the angle ω, with a negative angle leading to a thrust direction of -1, and a positive angle resulting in a thrust direction of 1. Interestingly, height y does not appear to significantly affect the engine direction.

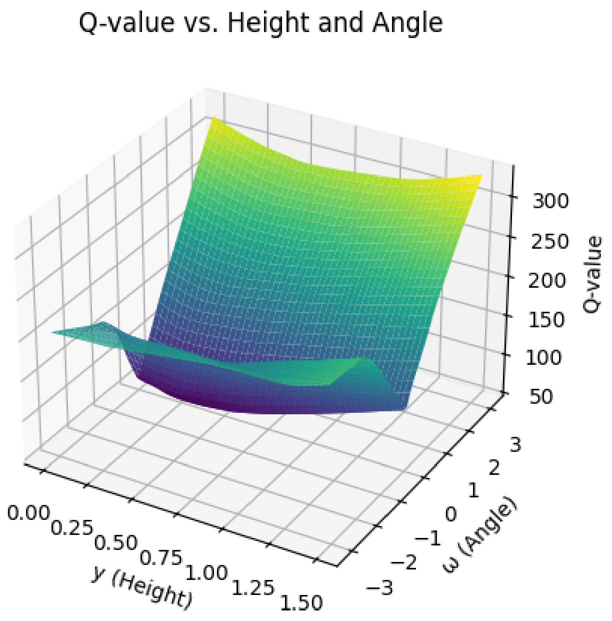


Figure 12: Q-value surface for varying y and ω.

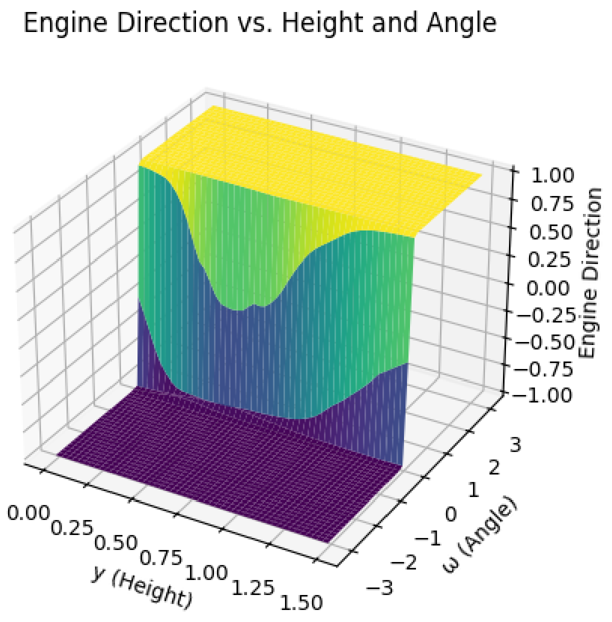


Figure 13: Engine direction $\pi_\theta(s(y, \omega))_2$ for varying y and ω.

Comparison of Agent Performance

We compared the performance of a random agent with that of the trained agent. The average reward for the random agent was -253.21, while the trained agent achieved an average reward of 211.55.

Figure 14 shows the total episodic reward over 50 episodes for both agents. As seen in the plot, both agents exhibit a similar pattern over time. The trained agent clearly outperforms the random agent, demonstrating the effectiveness of learning through experience in the reinforcement learning setup.

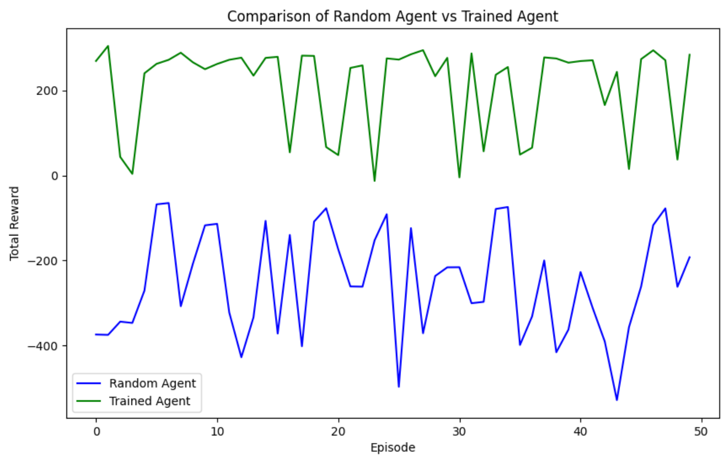


Figure 14: Comparison of Random and Trained Agent Rewards over 50 Episodes