Assignment 6

Dhruvi Mandaviya - 301299445

Matheus Vinicius Ferreira Figueiredo Teixeira - 301236904

# Discuss the rationale of the activation functions & the loss function used in the network. [10 points]

The implementation utilizes ReLU (Rectified Linear Unit) activation functions in the hidden layers of the neural network. This choice is particularly suitable for Deep Q-Learning as ReLU provides the necessary non-linearity while avoiding the vanishing gradient problem commonly encountered with sigmoid or tanh functions. For the output layer, no activation function is used, allowing the network to predict unbounded Q-values, which is essential as these values represent expected future rewards that can be both positive and negative in the LunarLander environment.

The loss function implemented is Mean Squared Error (MSE), calculated between the target Q-values and predicted Q-values. This choice is fundamental to Q-learning's objective of minimizing the difference between predicted and actual Q-values. MSE provides a smooth, differentiable loss landscape suitable for gradient descent optimization, and its quadratic nature appropriately penalizes larger errors more heavily. This is particularly important in the LunarLander environment where Q-values can vary significantly between successful landings (+100) and crashes (-100).

# Define the hyperparameters [50 points]

## the number of iterations

The training process employs a training interval of 4 steps, meaning the network updates its weights every fourth step within each episode. This interval represents a careful balance between learning frequency and computational efficiency. Training after every step would be computationally expensive and might lead to overfitting on recent experiences, while longer intervals could miss important learning opportunities. The training begins after 50 episodes, allowing the replay buffer to accumulate a diverse set of experiences necessary for stable learning.

## the number of episodes

The implementation uses 1000 episodes for training, a choice supported by empirical evidence in similar implementations of the LunarLander environment, such as the implementations by Chanseok Kang (Kang, 2021), Yu-Chen Chou (Chou, 2021), and baukesh (baukesh, 2021). This number provides sufficient time for the agent to explore the state space and develop effective landing strategies. The complexity of the landing task, which requires mastering multiple skills like attitude control and descent rate management, justifies this episode count.

## the maximum number of steps

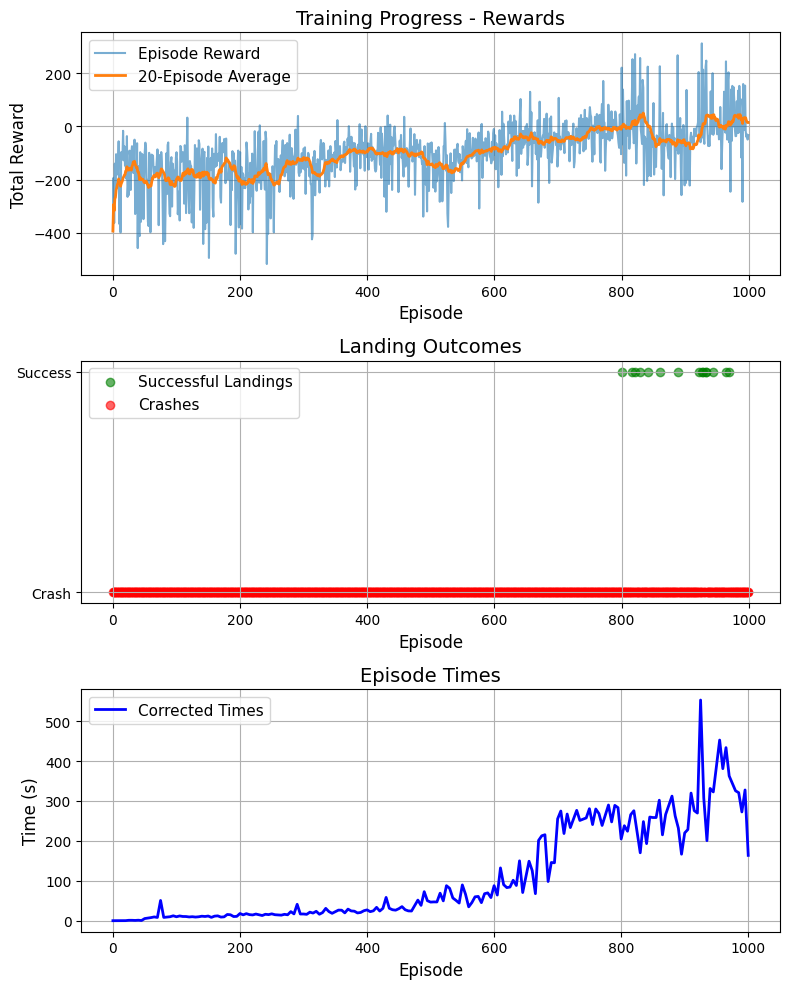
Each episode is limited to 1000 steps, adhering to the environment's built-in constraints. This limit provides sufficient time for the agent to execute complete landing sequences while preventing indefinitely long episodes. The step limit proved crucial in the training process, as it helped identify when the agent was stuck in suboptimal behaviors like continuous hovering.

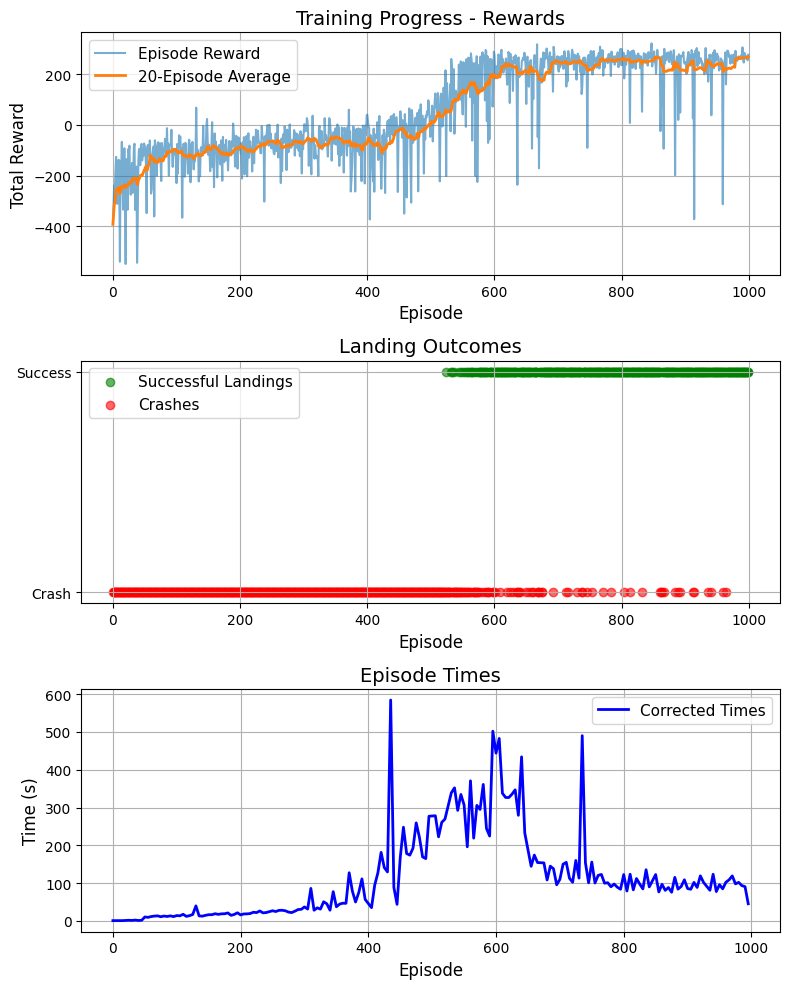
## the discount factor γ at each step.

A discount factor of 0.99 was chosen to strongly emphasize long-term rewards while maintaining numerical stability. This high value ensures the agent considers future consequences of its actions while still allowing effective backpropagation of rewards through the temporal sequence of decisions. A higher value might cause training instability, while a lower value could make the agent too focused on immediate rewards, potentially leading to suboptimal landing strategies.

# Analyze the agent's learning progress by plotting relevant performance metrics (e.g., cumulative rewards, episode length) over time. [10 points]

**First Version**

**Second Version**



**First Version**

Training Summary:

**Total training time:** 6h 12m 25s

**Episodes hitting max steps:** 229/1000

**Successful landings:** 15/1000 (1.5%)

**Crashes:** 985/1000 (98.5%)

**Best score achieved:** 312.91

**Second Version**

Training Summary:

**Total training time:** 6h 25m 21s

**Episodes hitting max steps:** 109/1000

**Successful landings:** 389/1000 (38.9%)

**Crashes:** 611/1000 (61.1%)

**Best score achieved:** 322.44

The learning progress of the agent was analyzed for two versions of the neural network implementation. Version 1 served as the initial attempt, but it exhibited significant challenges in achieving consistent positive rewards and solving the LunarLander environment. To address these issues, Version 2 was created with an enhanced architecture and adjusted hyperparameters. This section compares the performance of the two implementations, highlighting their cumulative rewards, episode lengths, and overall training efficiency.

In Version 1, the reward history revealed slow learning progress, with the agent only beginning to approach average rewards near zero after almost 700 episodes. Even then, the rewards were highly inconsistent, and the agent struggled to solve the environment with any reliability. Successful landings, defined as achieving a reward above 200, were rare, with most episodes resulting in crashes. The episode length data further supported these observations, showing that approximately 400 episodes ended very quickly, reflecting frequent crashes. Although the episode lengths began to increase after 400 episodes, the agent often became stuck in a hovering state, failing to make meaningful progress toward landing. While Version 1 did achieve a best score of 312.91, this was an isolated success rather than indicative of consistent performance.

Version 2 demonstrated significant improvements in both learning speed and stability. By increasing the number of neurons in each layer from 64 to 128, Version 2 provided the network with the capacity to learn more complex relationships between the agent's state and the optimal actions. This adjustment enabled the agent to achieve an average reward of +200 after only 600 episodes, 300 episodes earlier than in Version 1. The reward graph for Version 2 showed a clear plateau around 700 episodes, where the rolling average rewards stabilized near +200. This suggests that the agent had effectively learned the task by this point, indicating that an early stopping mechanism could be implemented to save computational resources. Episode lengths in Version 2 followed a different trajectory compared to Version 1. While early episodes were short due to frequent crashes, the lengths increased as the agent explored better strategies. Unlike Version 1, Version 2 eventually achieved shorter episode lengths even with successful landings, indicating that the agent had learned how to land efficiently without wasting time hovering. However, between episodes 425 and 625, the training was conducted on a local machine without GPU acceleration, contributing to increased computational time during this interval. This did not significantly impact the learning trajectory but is worth noting.

The decision to increase the number of neurons rather than the number of layers in Version 2 was particularly advantageous in this case. LunarLander has only eight input features, and the additional depth from more layers could have introduced unnecessary complexity, potentially leading to overfitting. By increasing the number of neurons in the existing layers, the network gained sufficient capacity to model the relationships needed for successful landings while avoiding the pitfalls of a deeper architecture.

# Discuss the challenges faced during training and potential strategies for further improving the agent's performance. [5 points]

The training process for solving the LunarLander environment presented several challenges related to both the learning dynamics of the agent and the limitations of the computational resources. The most significant challenge in Version 1 was the agent's inability to achieve consistent positive rewards. This was primarily due to the limited capacity of the neural network, which restricted its ability to learn complex control strategies for landing. To address this, Version 2 increased the number of neurons in each of the two hidden layers to 128. This adjustment enhanced the network's ability to learn nuanced patterns in the environment, leading to faster and more reliable learning. The choice to increase neurons rather than layers was particularly effective, as it balanced the model's complexity with its capacity to generalize, avoiding the risk of overfitting that might have arisen with a deeper architecture.

Computational resources posed another significant challenge during training. As the local machine lacked a GPU, a Google Cloud Platform (GCP) virtual machine was used to accelerate training. However, frequent connection issues with the VM interrupted the process, necessitating the implementation of a checkpointing system. By saving the model weights, training variables, and performance metrics every 25 episodes, the training process could be resumed seamlessly after interruptions.

The adjustments to the epsilon decay function were also critical in addressing the challenges of exploration and exploitation. While Version 1 decayed epsilon over 1000 episodes, Version 2 decayed epsilon over 700 episodes, allowing the agent to transition to exploitation more quickly. This adjustment was instrumental in helping the agent focus on refining successful strategies rather than continuing random exploration. The increased batch size of 128 in Version 2 also contributed to more stable updates during training, ensuring smoother learning progress.

In addition to these changes, the episode lengths provided valuable insights into the agent's performance. In Version 1, the agent often became stuck in hovering states, leading to long episode durations without successful landings. Version 2 not only reduced crashes but also optimized landing efficiency, as evidenced by the shorter episode lengths in later stages of training. This indicates that the agent learned how to achieve the goal efficiently, rather than merely avoiding failure.

While Version 2 demonstrated significant improvements, further optimizations could be explored. The addition of an early stopping mechanism could save computational resources by halting training once the performance plateau is reached, as observed around 700 episodes in Version 2. Gradient clipping could also be implemented to stabilize training by preventing large parameter updates. Finally, upgrading to a more powerful GPU or a dedicated cloud environment would enhance training efficiency and reduce interruptions.

In conclusion, Version 2 successfully addressed the limitations of Version 1 by improving the neural network architecture, adjusting the epsilon decay strategy, and implementing a robust checkpointing system. These changes enabled the agent to learn more effectively and efficiently, achieving consistent success in the LunarLander environment. Future improvements could build on these foundations to further enhance performance and training stability.

**References**

baukesh. (2021). *LunarLander-v2 with DQN*. Retrieved December 5, 2024, from <https://github.com/baukesh/lunar-lander-v2-dqn>

Chou, Y.-C. (2021). *DQN for LunarLander-v2*. Retrieved December 5, 2024, from <https://github.com/yuchen071/DQN-for-LunarLander-v2>

Kang, C. (2021, May 7). *Deep Q-Network (DQN) for LunarLander-v2*. Retrieved December 5, 2024, from <https://goodboychan.github.io/python/reinforcement_learning/pytorch/udacity/2021/05/07/DQN-LunarLander.html>