Final Project – Intro to AI (67842)

Lunar Lander

Team members:

Gal Kalimi

Yarin Ohayon

Ilan Kolker

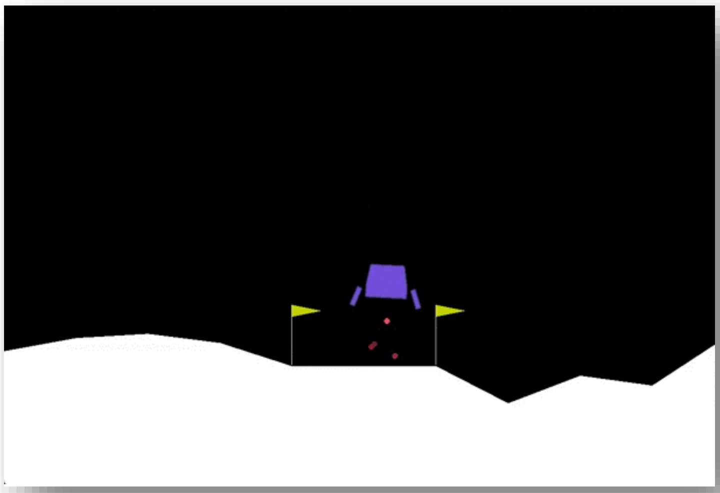
For access to the project code:
https://github.com/YarinOhayon/Lunar_Lander.git.

Contents

Introduction

"That's one small step for a man, one giant leap for mankind"
—Neil Armstrong

The Lunar Lander problem is a task that involves controlling a spacecraft as it attempts to land on the surface of the Moon. The primary objective is to safely land the spacecraft within a designated landing pad. This problem is challenging due to the dynamics of the lunar gravity, the need for precise control, and the changing of the surface. Our main goal is to direct the agent to the landing pad as softly and fuel efficiently as possible.



Solving the Lunar Lander problem is important for several reasons. It teaches us about control systems and autonomous navigation, which are essential for designing spacecraft landings and advanced robots. This problem simulates the challenge of landing on the Moon, helping us improve control strategies when dealing with limited fuel and needing precise movements. It's an interesting problem because it mimics real space missions, where accurate landings are crucial. With no Israeli spacecraft having landed on the Moon yet, this problem offers a chance for significant innovation.

Framework

The framework used for the Lunar Lander problem is Gymnasium, a toolkit developed by OpenAI for creating and comparing reinforcement learning algorithms. It provides various environments for testing and developing these algorithms. For our project, we use the Box2D simulator within the 'LunarLander-v2' environment.

Contributions

In addition to the framework and techniques, we received assistance from ChatGPT, an advanced language model. ChatGPT helped with:

- Designing the graphical user interface (GUI).
- Plotting graphs of the results.
- Refining the wording of certain sentences of our report.
- Assisting with some aspects of the PID implementation.
- Documenting the code.
- Creating and refining the README file.

Modeling of the world

Our environment models key aspects of the lunar landing scenario, including gravity, fuel consumption, and the impact of thruster controls. The spacecraft is equipped with a main engine and two side thrusters, which the agent can activate to control its vertical and horizontal movement. The objective is to land the spacecraft safely within a

designated landing zone while minimizing fuel usage and avoiding crashes. The environment provides continuous state feedback, including the lander's position, velocity, angle, and angular velocity. Each state in our environment will have 8 state variables associated with it:

- x coordinate of the lander.
- y coordinate of the lander.
- vx horizontal velocity.
- vy vertical velocity.
- θ orientation in space.
- ω angular velocity.
- Left leg touching the ground (Boolean).
- Right leg touching the ground (Boolean).

The discrete action space consists of predefined actions that the agent can take at each step:

1. firing the main engine.
2. firing the left thruster.
3. firing the right thruster.
4. doing nothing.

Our Approaches to the problem

We initially explored classical methods like planning and search algorithms. However, the continuous state space of the Lunar Lander made these approaches challenging. This led us to explore other approaches.

- **Proportional-Integral-Derivative (PID) Controller** - Control involves adjusting a system's inputs to achieve desired outcomes. In our project, we applied PID controllers to manage the Lander's vertical and horizontal movements, as well as its angle. By manually tuning the PID parameters, we established a baseline for the Lander's descent and landing performance.
- **Deep Q-Network (DQN) -** DQN is a popular reinforcement learning technique that uses deep learning to estimate the value of actions in different states. In our project, we customized the environment to include realistic fuel management constraints without altering the original reward system. We focused on improving the DQN agent's performance by fine-tuning various settings and exploring different strategies to balance efficiency with precise landing.

Previous Work

We have seen several attempts to solve this problem. The most common algorithms we encountered were RL algorithms. Each algorithm had its own strengths and weaknesses.

Algorithms

1. Classical Control Algorithms:

We saw some attempts to solve the Lunar Lander problem using classical control methods, such as PID controllers. While these methods provided a basic control mechanism, they were limited in their adaptability to varying conditions and complex dynamics. Some prior work explored adjusting setpoints for different control parameters to better handle specific landing scenarios. However, due to the simplicity of PID controllers, we chose to set all setpoints to zero. This decision allowed us to focus on the core control dynamics without extra complexity from multiple setpoints. We used this simplified PID controller as a baseline.

2. Q-Learning:

Q-Learning uses a Q-table to store and update the expected rewards for each action in each state. However, due to the continuous state space of the Lunar Lander, Q-Learning faces challenges such as state space explosion, making it impractical for larger or more complex versions of the problem. Solutions that tried applying this approach had to alter the state space and discretize it.

3. Deep Q-Networks (DQN):

Deep Q-Networks (DQN) represent a significant advancement in solving the Lunar Lander problem. By combining Q-Learning with deep neural networks, DQN effectively handles the continuous state space through Q-function approximation. This method has proven superior to traditional Q-Learning, particularly in complex and large state spaces. DQN gained prominence through the 2013 paper *Playing Atari with Deep Reinforcement Learning* by DeepMind's researchers.

4. Policy Gradient Methods:

Policy Gradient methods, such as REINFORCE, directly optimize the policy by following the gradient of the expected reward. These methods have been applied to the Lunar Lander problem with success, especially when combined with function approximation. However, they often require careful tuning of hyperparameters and are sometimes less stable than value-based methods like DQN.

Performance Benchmarks

The performance of these algorithms is typically measured by the average reward per episode, where a higher score indicates a more successful landing. According to the official Gymnasium documentation, an episode is considered a solution if it scores at least 200 points. We saw that a well-trained model can achieve an average reward exceeding 200, which corresponds to a successful landing in most cases. While working, we kept assessing our solution to see how we were doing compared to the desired 200 points goal.

Common Assumptions

Discrete Action Space

Many implementations assume a discrete action space, which simplifies the learning process but may not capture the full range of possible actions available to a lunar lander.

Simplified Dynamics

The problem assumes simplified dynamics, such as a 2D environment and uniform gravitational forces. While these assumptions make the problem more tractable, they may not fully represent real-world lunar landing scenarios.

Reward Structure

The reward structure typically penalizes fuel usage and harsh landings while rewarding smooth landings and remaining upright. These rewards are often hand-tuned to encourage specific behaviors.

Methodology

Proportional-Integral-Derivative (PID) Controller (Baseline model)

With the help of our tutor, we discovered the concept of Control and implemented PID controllers as a baseline solution for our Lunar Lander project.

Description of the Model

In control systems, the "plant" refers to the system being controlled—in our case, the Lunar Lander. The PID controller adjusts the inputs (thrust activations) to the plant based on feedback from the system (state observations) to achieve a desired outcome (safe landing).

The PID controller operates by calculating an error value, denoted as $e(t)$, as the difference between a measured value (the current state of the lander) and a desired setpoint (the target position or orientation). It then uses three terms to compute the control input:

Proportional (P) Term:

$$P(t) = K_p \cdot e(t)$$

Produces an output value proportional to the current error value. The proportional response is controlled by the constant K_p.

Integral (I) Term:

$$I(t) = K_i \cdot \int_0^t e(\tau)d\tau$$

Accounts for past errors by integrating them over time, using the timestamp dt. It helps to eliminate any residual steady-state error that the proportional term alone cannot address. The integral response is controlled by K_i.

Derivative (D) Term:

$$D(t) = K_d \cdot \frac{d}{dt}e(t)$$

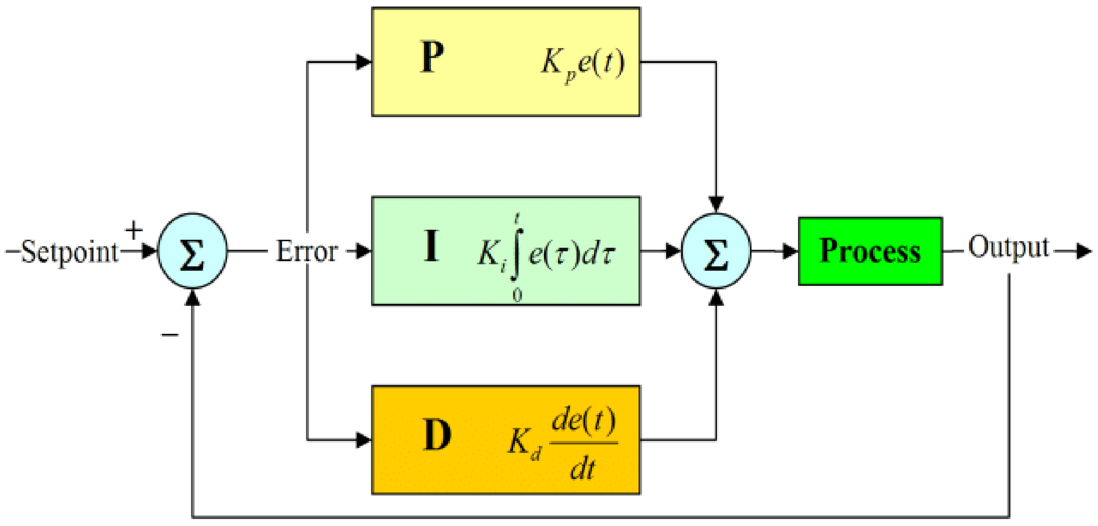
This term predicts future errors based on how quickly the error is changing over the timestamp dt. It helps to smooth out the system's response and make it more stable. The derivative response is controlled by K_d.

The output of the PID controller $u(t)$ is the sum of these three terms:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{d}{dt}e(t)$$

This output is applied as the control input to the system as shown in the diagram below, adjusting its behavior to minimize the error and guide the system towards the desired setpoint.

An Illustration of PID Controller



We implemented three PID controllers to manage different aspects of the lunar lander's behavior:

Vertical PID Controller: Controls the lander’s vertical movement, aiming to bring it smoothly to the ground. The setpoint is set to zero, meaning we want to control the descent speed to prevent hard landings.

Horizontal PID Controller: Manages the lander’s horizontal movement to keep it centered. The setpoint is also zero, ensuring the lander stays on target without drifting sideways.

Angle PID Controller: Keeps the lander upright during descent. The setpoint is zero, aiming to maintain the lander’s angle close to vertical, preventing it from tipping over.

Various methods were explored for tuning the PID parameters, including grid search, but we encountered limitations, such as the lander getting stuck during tuning. Ultimately, manual tuning was chosen, where the best average reward from several iterations was taken to calibrate the controller effectively.

Assumptions made

- State Feedback and Timing (dt): The controller assumes that state feedback is provided with a constant, accurate time step (dt). In practice, any inconsistencies in dt could affect the performance of the PID controller, as it relies on precise timing to compute integral and derivative terms.
- Symmetry: The controllers are designed based on the assumption that the lander's vertical and horizontal movements can be controlled independently, treating them as separate systems. However, there may be interactions between the two, where changes in one direction could influence the other.

Success Criteria

The success of the PID controller is evaluated based on **error reduction**. The PID controller's performance is graded based on its ability to minimize the error, which represents the difference between the lander's current state and its desired setpoint.

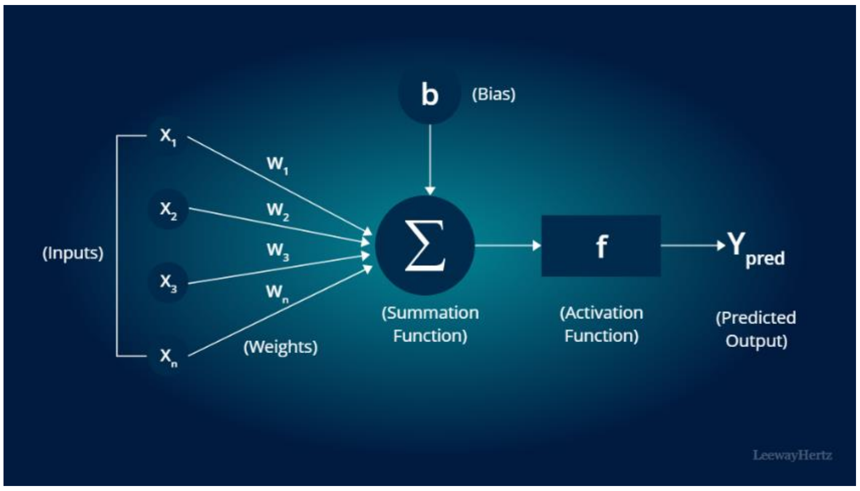
Deep Q-Network (DQN) (Primary Model)

To enhance the Lunar Lander's performance, we implemented a Deep Q-Network (DQN). This model leverages deep learning to address the complexities of the lunar landing environment. Before diving into our DQN model let's understand the concept of Neural Networks.

Neural Networks Overview

Neural networks are computational models inspired by the human brain, consisting of interconnected layers of nodes (neurons). These networks can learn to map inputs to outputs by adjusting the connections (weights) between neurons based on the data they process.

An illustration of a Neural Network



Description of the Model

DQN is an advanced reinforcement learning approach that combines Q-Learning with deep neural networks. The network approximates the Q-value function, which estimates the expected future rewards for each action given a particular state. This is particularly useful for environments with large or continuous state spaces, such as our Lunar Lander scenario.

Network Architecture

The DQN is implemented using PyTorch and consists of the following layers:

- **Input Layer**: Accepts state vectors with a dimensionality of state_dim.
- **Hidden Layers**:
  - Three fully connected layers with 256, 256, and 128 neurons, respectively. These layers use ReLU activation functions.
- **Output Layer**: Produces Q-values for each action in the action space.

The model's forward pass computes Q-values based on the input state, which are used to determine the best action to take in the given state.

Training the DQN

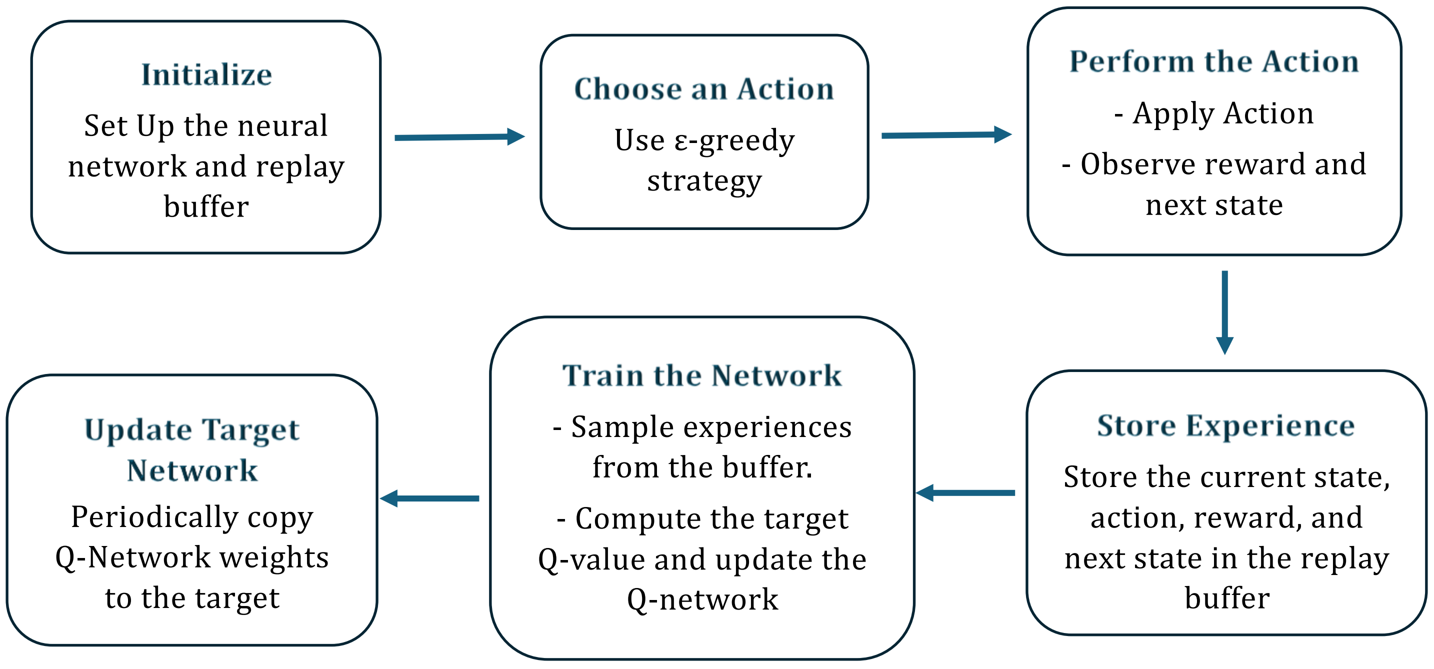
Training the DQN involves the following steps:

1. **Experience Replay**: We store past experiences (state, action, reward, next state) in a replay buffer. During training, we randomly sample batches from this buffer to break the correlation between consecutive experiences.
2. **Target Network**: We use a separate target network to stabilize training. The target network's weights are periodically updated with the weights of the main network.
3. **Loss Function**: The network is trained to minimize the mean squared error (MSE) loss between the predicted Q-values and the target Q-values.
   For each experience $\langle s_t, a_t, r_t, s_{t+1} \rangle$, we compute the target Q-value using modified form of Bellman equation. This target value represents the expected future reward for taking action a_t in state s_t, and is calculated as:

   $$Q_{target}(s_t, a_t) = r_t + \gamma max_{a'} Q(s_{t+1}, a')$$

   Where r_t is the immediate reward received after taking action a_t in state s_t, s_{t+1} is the next state and $\gamma \in [0,1]$ is the discount factor.
4. **Optimization**: We use an optimizer, such as adaptive moment estimation (Adam), to update the network weights based on the computed loss.

A scheme of the training procedure


Initialize
Set Up the neural network and replay buffer
Choose an Action
Use ε-greedy strategy
Perform the Action
- Apply Action
- Observe reward and next state
Store Experience
Store the current state, action, reward, and next state in the replay buffer
Train the Network
- Sample experiences from the buffer.
- Compute the target Q-value and update the Q-network
Update Target Network
Periodically copy Q-Network weights to the target

Success Criteria

The success of the DQN model is determined by the **reward function**, which serves as the core criterion for evaluating the agent's performance. The reward function includes several components:

Proximity to Target: Rewards for landing close to the designated spot between the flags.
Touchdown Velocity: Penalties for high velocity at touchdown, encouraging gentle landings.
Fuel Efficiency: Rewards for minimizing fuel consumption, promoting efficient maneuvers.
Landing Angle: Reward or penalize based on the angle of the lander at touchdown to ensure the lander lands upright.

Reward Function Formula

The reward function is designed to encourage successful landings. The terms Shaping and Previous Shaping refer to the points that the current and previous state are granted. The subtraction between the current shaping and the previous shaping evaluates how much better is the current state compared to the previous one. Then, we subtract points from this evaluation for engine usage, to reward fuel efficient models.

The Reward formula is given by:

$$Reward = (Shaping - Previous\ Shaping)$$
$$-0.30 \cdot MainEnginePower - 0.03 \cdot SideEnginePower$$

while Shaping is given by:

$$Shaping = -100 \cdot \sqrt{\left(position^2{}_x + position^2{}_y\right)}$$

$$-100 \cdot \sqrt{\left(linearVelocity^2{}_x + linearVelocity^2{}_y\right)}$$

$$-100 \cdot |angle|$$

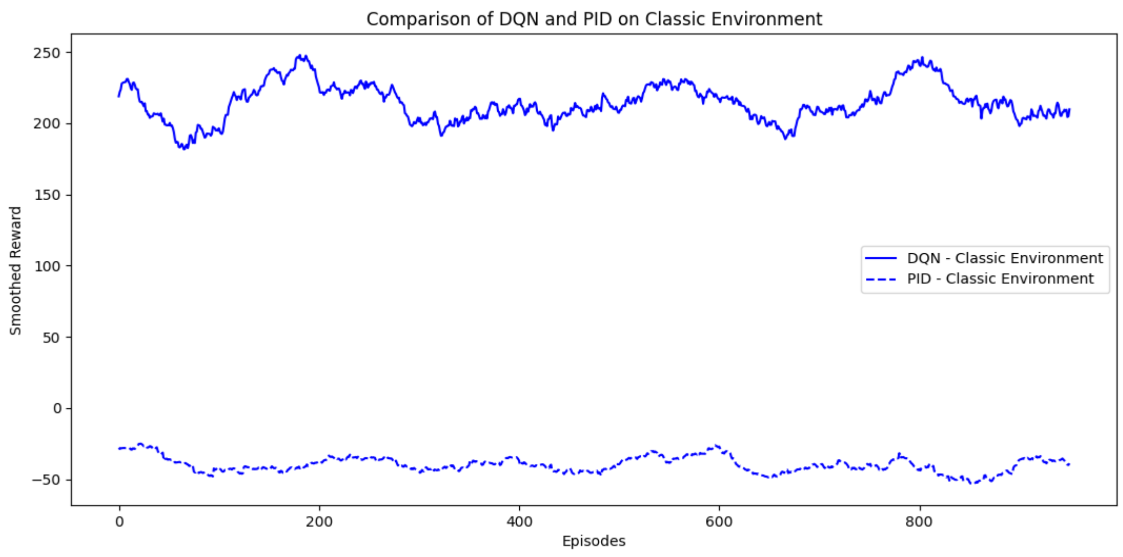
$$+10 \cdot legContact_1 + 10 \cdot legContact_2$$

Results

In this section, we present the findings from our experiments with the Lunar Lander problem using the algorithms we described above. Our primary model, the Deep Q-Network (DQN), was evaluated against a baseline model - PID, to establish performance benchmarks.

Analysis of DQN vs. PID Performance on the Classic Environment

The following graph compares the performance of a DQN model with a PID controller in the classic Lunar Lander environment.



DQN Model Performance

The DQN model consistently achieves high rewards, with values mostly ranging between 200 and 250. This indicates that the model has effectively learned the optimal policy for landing the spacecraft in the classic environment.

The stability of the DQN's performance, with relatively minor fluctuations, suggests that the model is well-suited to handle the classic environment's dynamics. It can manage fuel consumption, thrust, and descent, leading to successful landings with high rewards.

PID Controller Performance

The PID controller, on the other hand, struggles significantly in the same environment. Its rewards mostly hover around -40. This negative reward indicates frequent crashes or inefficient landings, where the spacecraft either uses too much fuel or fails to land safely.

The PID controller's poor performance highlights its limitations in this environment. Unlike the DQN, which learns and adapts to the environment through training, the PID controller relies on predefined proportional, integral, and derivative gains, which may not be well-suited to the complexities of the Lunar Lander environment.

Overall Comparison

The DQN model has an advantage over the PID controller due to its ability to learn from experience and adapt to the environment's details. This adaptability is evident in the DQN model's consistently higher rewards, even though it exhibits slightly larger fluctuations. In contrast, the PID controller, despite having smaller fluctuations, performs significantly worse overall. The PID controller's inability to adjust to the environment's complexities limits its effectiveness. While it serves as a useful starting point, the PID controller's weaker performance highlights the importance of using more advanced approaches like DQN for complex tasks.

Conclusions

The comparison clearly illustrates the superiority of the DQN model in the classic Lunar Lander environment. The DQN's ability to learn the classic environment makes it far more effective than the PID controller, which struggles to achieve even moderate success. This analysis reinforces the value of reinforcement learning approaches like DQN in environments where traditional control methods fall short.

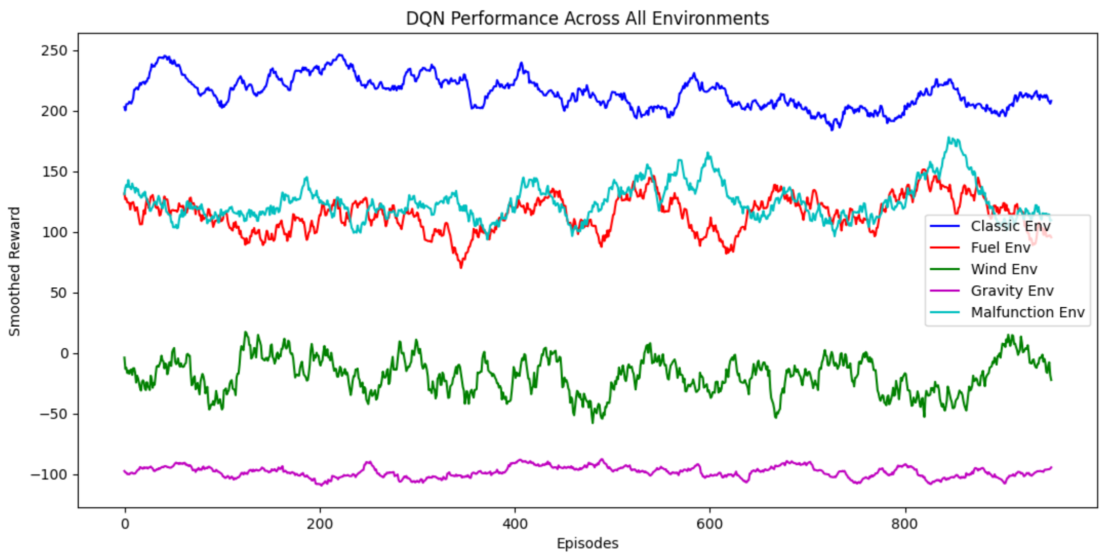
Analysis of DQN Performance Across Different Environments

The DQN model was trained in the classic environment and evaluated in five different environments to explore its adaptability and robustness. After this short explanation on each environment, we will show the graph of the DQN performance over those environments.

1. **The Classic Environment:** The standard Gymnasium environment without any modifications. This scenario follows the classic Lunar Lander problem setup and serves as a benchmark to evaluate the agent's performance under normal conditions.
2. **Fuel-Constrained Environment:** In this variation, the lander is provided with a limited fuel supply. Once the fuel is depleted, the engine cannot be used, requiring the agent to optimize its movements carefully to achieve a soft landing.
3. **Engine Malfunction Environment:** This environment introduces random engine malfunctions that temporarily disable the engine, making this variation a

test of the agent's ability to handle unpredictability and recover from system failures.

4. **High Gravity Environment:** In this scenario, the gravitational force is doubled from the classic one, simulating different planetary conditions. The agent must adjust its landing strategy dynamically to cope with changes in gravity, making it a robust test for generalizing across varying gravitation.
5. **Variable Wind Environment:** This environment includes random wind gusts that can push the lander off course. The agent must continuously adjust its landing trajectory to counteract the effects of the wind, testing its ability to maintain control and stability in the face of external disturbances.



Classic Environment (Blue Line)

The DQN model in the Classic Environment achieves the highest rewards, consistently performing well across all episodes. The reward curve stabilizes around a high value (above 200), indicating that the agent has learned well and produces good results for the standard Lunar Lander environment. This result serves as a benchmark, showing the agent's ability to learn and optimize its actions under predictable conditions without additional constraints or interruptions.

Fuel-Constrained Environment (Red Line)

When the DQN trained in the Classic Environment is evaluated in the Fuel-Constrained Environment, its performance drops, with rewards averaging between 100 and 150. The agent shows some adaptability but struggles with the new constraint. Since the DQN was not trained to handle a fuel limit, it does not fully understand the need to conserve fuel. The drop in performance indicates that while the agent can still land the lander, it does not optimize fuel usage effectively, leading to suboptimal actions once the fuel is low or depleted. The performance drop reflects the fact that the DQN's learned policy from the Classic Environment is not directly transferable to this new constraint.

Engine Malfunction Environment - (Cyan Line)

The malfunctioning environment (cyan line) shows the DQN achieving rewards around 100-150. Despite the engine failing 10% of the time, the model still maintains relatively good performance, indicating a surprising resilience to sporadic engine failures. However, the instability and variance in the rewards suggest that these malfunctions do introduce a level of unpredictability that the model doesn't fully overcome.

High Gravity Environment (Purple Line)

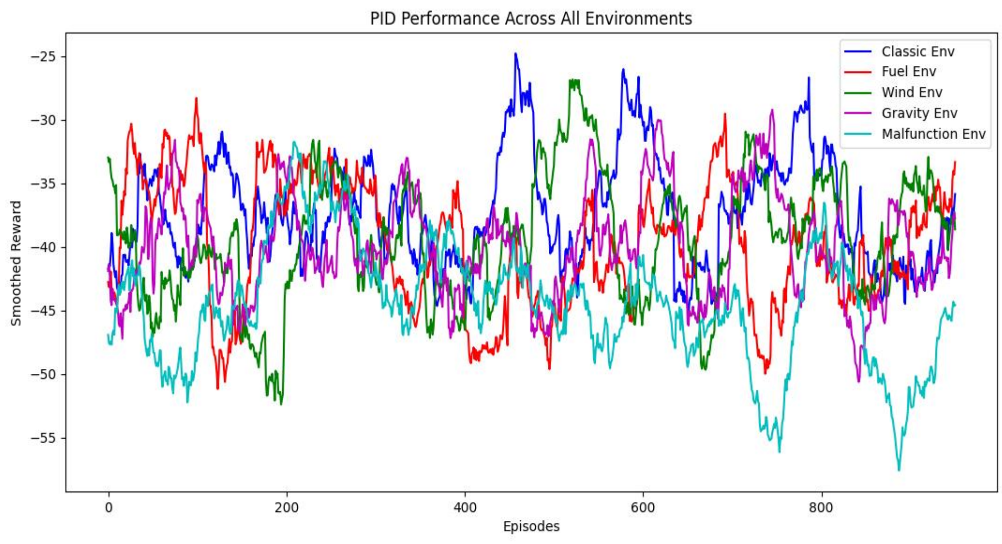
The DQN performs the worst in the Gravity Environment, where rewards are consistently negative (around -100). This suggests that the agent, trained under fixed gravity, cannot cope with the varying gravitational forces. Gravity changes fundamentally alter the dynamics of the environment. Since the DQN was trained under a single, consistent gravitational force, it does not know how to adapt to the variations, leading to poor landing performance and consistently negative rewards. We can infer that the learned policy from the classic environment becomes useless in this environment. The network failed to generalize to the new physics, resulting in the worst performance among all environments.

Variable Wind Environment (Green Line)

In the Wind Environment, the rewards are hovering around 0. This indicates that the DQN, trained without considering wind disturbances, struggles to adapt to the unpredictable forces. The introduction of wind, a stochastic disturbance, significantly disrupts the agent's learned policy. Since the DQN was not trained with this variability, it cannot anticipate or react effectively to the random shifts caused by the wind, leading to poor and inconsistent performance. The architecture that worked well in the predictable Classic Environment fails to generalize to an environment with such high variability.

Comparison to PID performance

Let's look at the performance of PID in the environments mentioned above:



As we can see, the PID rewards on the new environments were very similar to those achieved on the classic environment. All the rewards on those environments for 1000 iterations are between -25 and -60. This means that the PID model performances were consistent and did not seem to be affected that much by the environment changes, as opposed to the DQN model. This can be explained by the fact that the PID controller operates based on predefined control rules that are less sensitive to changes in the environment dynamics.

In contrast, the DQN model shows more varied performance across different environments. It performed well on the classic environment and did relatively well on malfunction and fuel environments. However, it struggled significantly on environments with increased complexity, such as wind and increased gravity. This is likely because the DQN relies on learning optimal policies from interactions with the environment, and when the dynamics change, its learned policy may no longer be as effective. This suggests that while the DQN is more adaptable in theory, it requires more sophisticated training techniques to generalize better across multiple scenarios, whereas the PID controller performs more uniformly but with lower overall rewards.

Overall Observations

- **Robustness and Adaptability**: The DQN model demonstrates strong robustness in environments that closely resemble the training environment (classic and fuel-restricted). However, as the level of environmental complexity and randomness increases (wind, high gravity, malfunction), the model's performance degrades, highlighting areas where the DQN might benefit from further training or algorithmic adjustments.
- **Environmental Complexity**: The results underscore the importance of training reinforcement learning models in diverse environments. A model trained solely on a classic environment may not generalize well to more complex or varied conditions, as seen in the significant performance drop in the malfunctioning and high-gravity environments.
- **Future Directions**: To improve the model's performance across all environments, future work could involve training the DQN model in multiple environments simultaneously or employing techniques like domain randomization. Additionally, enhancing the model's ability to handle stochastic events, such as engine malfunctions, could be achieved by incorporating elements of robust reinforcement learning or by fine-tuning the model with more varied scenarios.

Attempts for Improvements

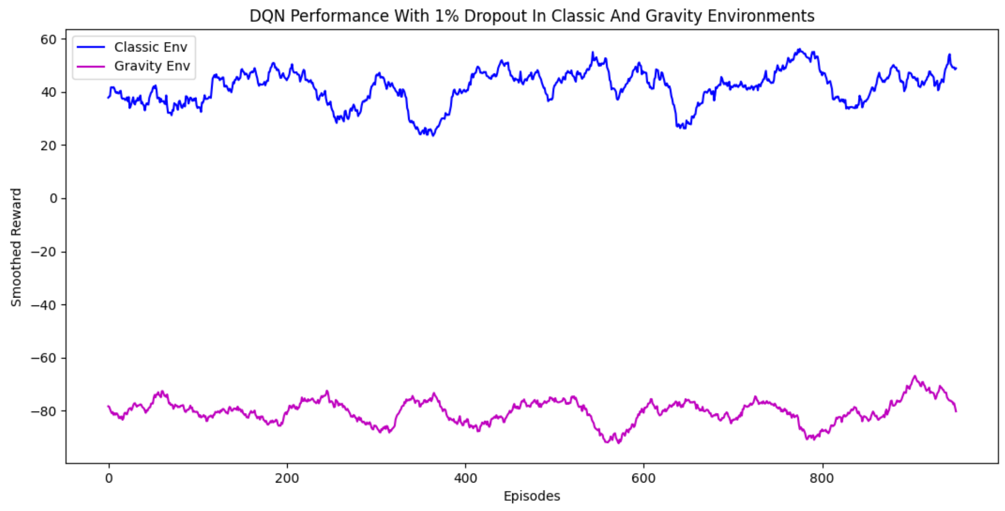
As stated above, one of our main observations was the fact that the DQN model didn’t perform well in an environment with higher gravity. We decided to take it as a challenge and try to achieve a model that would be able to be trained on the classic environment and will perform better on an environment with twice as much gravity.

First, we tried fine tuning the hyper-parameters, aspiring to adjust the learning process in a way that would let it generalize more. We tried many different combinations of values for hyper-parameters such as learning rate, epsilon decay and the number of iterations. Those tries resulted in the same or even worse performances than those achieved by our initial DQN model.

Next, we explored different paths. It became clear that to allow our model to generalize better, we had to reduce overfitting. So, we implemented a modified version of our DQN model with an optional dropout layer positioned between the first hidden and the second hidden layer.

Initially we started with high dropout probability to reduce overfitting and improve generalization. However, this resulted in models that performed worse on both the classic and high-gravity environments. We then lowered the dropout probability, hoping to improve performance.

After several tries and tests, we ended up with a model that managed to do better in the high-gravity environment. The following graph shows the performance of a DQN model with 1% dropout probability on both environments.



First, we see a decent decrease in the model’s average rewards in the classic environment. This result is reasonable since we tried to generalize and reduce overfitting, so the scores on the training environment were expected to decrease. We can also see that this model averages rewards around -80, an improvement of 20+ points in comparison to the classic DQN model.

Summary

In this project, we addressed the Lunar Lander problem using two models: a baseline PID controller and a DQN model, both in the Gymnasium environment. The goal was to compare these approaches in terms of performance and adaptability.

The Lunar Lander problem involves controlling a spacecraft to land on a target using its engines while dealing with forces like gravity and thrust. The task requires precise control, balancing fuel use, and avoiding crashes. We implemented two models for this: a PID controller and a DQN model.

The PID controller is a simple, rule-based approach. It adjusts the lander's engines based on its position, angle and velocity, providing a control signal proportional to the error between the current and desired state.

The PID controller performed poorly in the Lunar Lander environment. Despite attempts to fine-tune the parameters, the results were consistently below expectations. The poor performance of the PID controller can be attributed to several factors, including some related to our implementation:

- **Model’s Assumptions:** The model assumes independent control of vertical and horizontal movements, which may not fully account for interactions between these dimensions. Additionally, the constant time step (dt) assumption may lead to performance issues if timing inconsistencies arise.
- **Complexity of Environment:** The PID controller's fixed gains are not well-suited for the dynamic and complex nature of the Lunar Lander environment. The PID controller lacks the ability to adjust its parameters dynamically.
- **Manual Tuning for Hyperparameters:** Manual tuning allowed for some adjustments but was time-consuming and may not have achieved optimal settings. We also tried automatic methods like grid search but faced issues such as the Lunar Lander getting stuck or performing poorly. This suggests we might have missed better hyperparameters. Exploring advanced or combined tuning techniques could potentially improve performance.
- **Choice of Setpoints**: We opted to set all PID setpoints to zero to simplify the control process. However, this decision may have limited the controller’s effectiveness. As observed in some previous works, using different setpoints might have enhanced the PID controller’s performance by providing more precise control under varying conditions.

The **DQN**, which learns optimal actions through reinforcement learning, was a more flexible solution. It takes the state of the lander and outputs the best action to minimize the error over time. We initially trained it on the standard environment and later added a dropout layer to improve robustness.

The DQN model was able to learn effective policies and adapt to the standard environment. When tested on modified environments that introduce fuel limitations, wind and engine malfunction the model performed relatively well. However, in environments with high gravity, the model struggled to generalize. We then Introduced dropout was trying to reduce overfitting and improve generalization, but the improvement was minimal.

The DQN's struggles in other environments can be traced to several key factors:

- **Model Capacity and Architecture:** It is possible that our DQN model's architecture is too simple. It may not have enough capacity to learn complex relationships that would help it perform better in the classic environment and generalize to new environments.
- **The Training Process:** We aimed to improve the model's performance in a high-gravity environment. However, it is important to note that we attempted this while training exclusively on the classic environment. This might have been too ambitious for a DQN model. It would be interesting to explore how models trained on a variety of environments would perform. On one hand, we could expect them to better adapt to the challenges of new environments. On the other hand, training on different environments might make it too difficult for the model to effectively land the spacecraft in any environment.

Another interesting aspect to consider is how these models would perform under non-discrete action spaces. This transition could make it more challenging for the DQN model to converge to an optimal policy and would likely require different exploration techniques.

The PID controller, on the other hand, could perform better as it is naturally suited for continuous control, but its lack of learning capability would still limit its adaptability to complex dynamics compared to reinforcement learning methods.

This project provided valuable learning opportunities, and we hope it has also highlighted engaging topics that encourage further exploration of the subjects discussed.